AD-A045 031 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2

AN EXPERIMENT IN SOFTWARE ERROR OCCURRENCE AND DETECTION.(U)

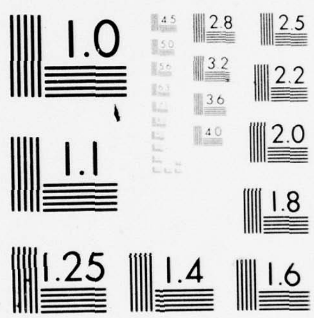
JUN 77 H HOFFMANN

UNCLASSIFIED NL

1 OF 4

AD

A045031



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

An Experiment in
Software Error Occurrence and Detection

by

Heinz-Michael Hoffmann

June 1977

Thesis Advisor: N. F. Schneidewind

Approved for public release; distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 An Experiment in Software Error Occurrence and Detection.	9	5. TYPE OF REPORT & PERIOD COVERED Master's Thesis, June 1977
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) 10 Heinz-Michael Hoffmann		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940	11	12. ~~REPORT DATE~~ June 1977
		13. NUMBER OF PAGES ~~319~~ 12 318p.
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Error Occurrence, Software Error Detection,
Experiment in Software Error Occurrence
Experiment in Software Error Detection

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The occurrence and detection of software errors was studied in four software projects. Errors were analyzed with respect to complexity measures of individual subroutines.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

This paper also provides definitions of error categories and types. The detailed documentation of the software production effort allowed both a qualitative and quantitative analysis of software errors with respects to error types.

Approved for public release; distribution unlimited

AN EXPERIMENT
IN
SOFTWARE ERROR OCCURRENCE AND DETECTION

by

HEINZ-MICHAEL HOFFMANN
LDCR, FEDERAL GERMAN NAVY

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
JUNE 1977

Author ____Heinz-Michael Hoffmann____

Approved by : ____N. F. Schneidewind____
Thesis Advisor

____Gary M Raetz____
Second Reader

____
Chairman, Department of Computer Science

____H Shrady____
Dean of Information and Policy Sciences

ABSTRACT

The occurrence and detection of software errors was studied in four software projects. Errors were analyzed with respect to complexity measures of individual subroutines. This paper also provides definitions of error categories and types. The detailed documentation of the software production effort allowed both a qualitative and quantitative analysis of software errors with respect to error types.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

The continued support and critical help provided by my thesis advisor Prof. N. F. Schneidewind, the contribution of LT. G. M. Raetz towards the completion of this report and the continuing patience of my wife Helga during this research period are gratefully acknowledged.

I. INTRODUCTION

In a recent SHARE study the annual growth rate of software demands over the years 1957 - 1985 was estimated to be as high as 21-23% [15]. The same study indicates that software production may develop at a growth rate of only 11.5-17%.

Besides this quantitative gap, Boehm [7] points out the following qualitative deficiencies: Since software is increasingly used for important functions such as defense systems, traffic control systems, and medical purposes, software is needed which can be trusted. Although 80% of the money used for the production of the Apollo Manned Spaceflight Program was devoted to testing [12], software errors occurred in Apollo 8, 11, and 14. One critical error even occurred during the lunar landing phase.

Software engineering (for definition see Chapter II) may be accepted as the method which yields the answer to the problem of producing reliable software at reasonable cost [7]. Little is known about software errors besides their existence and their negative influence on cost and performance. Increasing difficulties to produce sufficient amounts of high quality software make it mandatory to invent software development methods which are designed to avoid the most common software errors, achieve easier testing and

debugging and improve methods of error detection.

This cannot be done without detailed studies about the occurrence of software errors. Only this knowledge will help to identify the nature of errors and finally lead to improvements of software methodology. Other motivating factors for this study were:

1. Lack of currently available data to support analysis and research in the area of error ocurrence and detection.
2. The need to examine existing software error models.

The objective of the experiment was to gather accurate and complete information about errors, their occurrence and detection throughout the entire process of software development. Limitations of the experiment were the relatively small number of projects and the lack of experienced programmers. The use of the experimenter as the subject programmer and evaluator introduced biases also. The scope of the experiment did not include the program maintenance phase. It was felt to be of great importance to record all errors, even trivial errors which could hurt the image of the programmer. A detailed description, of the experiment and the error data is given in chapters 4 thru 5 and in the appendices.

II. CONCEPTS AND KEYWORDS

Communication among computer scientists is hindered by the lack of common terminology. In order to avoid misunderstanding, the most important keywords will be defined in this chapter. Wherever possible definitions are adopted from other publications.

A. SOFTWARE ENGINEERING TERMS

1. Software Engineering

"The practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate and maintain them". [7]

2. Top-Down Design

A program design method which starts at a very general level by identifying major functions and proceeds stepwise to lower levels to the identification of lesser functions that derive from the major ones. "Top-down design" and "hierarchical program design" are equivalent terms. [12]

3. Structured Programming

Although this term is used in numerous publications its understanding varies because of its conceptual nature. As proposed by Dijkstra as early as 1965 [30], it first was a concept to eliminate GO TO statements. Later on it was combined with the idea of top-down design. [31]

Rather than providing a formal definition, Yourdan [12] and Auerbach [27] describe structured programming by its objectives: "increase of readability" and "decrease of testing problems." The following definition of structured programming as it applies to this paper is derived from Kirchgaessner's description. [28]

Structured programming is a method of developing programs for which the underlying concept is a top-down design which eventually leads to a modular structure of the program. All algorithms are designed using the five major constructs allowed in structured programming:

- Sequential statement
- IF THEN ELSE statement
- WHILE loop (conditional loop)
- FOR loop (iterative loop)
- CASE statement

In addition to this definition it should be

emphasized that structured programming is a design discipline rather than a way of writing code. Therefore it should not be defined as "coding without GO TO statements." Methods have been developed to implement structured designs in FORTRAN. [12] However, it is easier, and more readable to use a block structured language, such as ALGOL, to implement a structured design.

4. Directed Graph

"A directed graph is a geometric graph, consisting of nodes and arcs with a direction of traversal associated with each arc." [16]

5. Module

"A module is a physical combination of program instructions that is independent of others with respect to compiling, assembling and loading and which performs a specific function." [16] "Each module has a small number of interactions with other modules." [4]

6. Program

A program is a set of integrated modules. [16]

7. Software Development Process

The software development process includes the

following stages:

- Problem Analysis
- Design and Design Review
- Coding
- Debugging
- Testing
- Integration
- Implementation
- Maintenance

Despite minor differences with other publications such as Auerbach [2] the above view of the software development process will be adopted as the definition of the software development process for this thesis. Although the development stages are listed sequentially it should be noted that software production in general is viewed as a dynamic process including feedbacks to earlier stages of a project. Thus none of these phases may be considered to be completed when the subsequent phase has started. In some projects, testing itself is preceded by activities such as the writing of test specifications and test procedures. Figure 1 indicates possible feedback situations within a complex program development process.

```
ANALYSIS
o
:
:            o..←..............................o..←.........o
:            :                                 :            :
:            :                                 ↑            ↑
:  ..←.......o..←......                        :            :
:  :         :          :                      :            :
v  :         :          ↑ Coding   Debugging   :            :
:  :         :          o-->------o-->---------o            :
:  : System  : Program  :                      :            :
o->o-->------o-->-------o                      :            :
   : Design  : Design                          :Integration :
   :         :                                 o-->---------o->o
   ↑         :                                 : Testing    :  :
   :         : Writing of Test  Test Design    :            :  :
   :         o-->---------------o-->-----------o            :  v
   :         : Specifications   :              :            :  :
   :         ↑                  ↑              :            :  :
   :         :                  :              :            :  :
   :.........o..←...............o..←...........o..←.........:  :
                                                               :
                                              IMPLEMENTATION   o
```

FIGURE 1

THE FEEDBACK CYCLES DURING A SOFTWARE DEVELOPMENT PROJECT

8. Common Data

Common Data is data which is used by several modules.

B. TESTING, DEBUGGING AND ERRORS

1. Debugging

(a) Debugging is the action to check subroutines and modules as to whether or not they perform according to programmer expectations. It is also the action which one takes to locate and correct known errors.

(b) Testing is the action taken after modules have been integrated to check whether or not the program meets specifications. As stated by Dijkstra [29], testing can only establish the presence of errors, not their absence.

2. Software Error

A software error is a mistake made during software development, which leads to an incorrect action or result with respect to the program specifications within the program. [16] Errors made in the specification phase of the program, hardware errors and compiler errors (i.e. errors within the compiler program) were not studied during this experiment. Categorization and definitions of software errors with respect to their occurrence are given in chapter 3.

C. SOFTWARE RELIABILITY AND CORRECTNESS

1. Software Reliability

"Software reliability is the probability that a computer program will perform its (user) intended function for a specified time interval under stated operating conditions." [17]

2. Software Correctness

"The feature of software that renders it operationally useful for its intended functions is its correctness. Correct software does the things intended by design." [9]

III. DEFINITION OF ERROR CATEGORIES AND TYPES

When analyzing the causes of errors in software production there is an obvious need to precisely describe an error in accordance with its category and type. One of the objectives of this study was to identify errors with respect to their occurrences and causes.

It was was found that many errors in software production belong to one of the following major categories:

1. Problem Specification Errors
2. System Design Errors
3. Program Design Errors
4. Coding Errors
5. Clerical Errors
6. Debugging Errors
7. Testing Errors
8. Implementation Errors

A. DEFINITONS OF ERROR CATEGORIES

1. Problem Specification Error

Any mistake or deficiency which occurs in the analysis of the program application or in specifying the software requirements with respect to the intended program application such as incomplete, erroneous or

ambiguous statements.

2. System Design Error

An error made in the transformation of the user-originated program specifications into systems design specifications.

3. Program Design Error

A program design error is one made in the transformation of the system design into specific algorithms and data structures.

4. Coding Error

A coding error is one which is made during the transformation of a program design into source language.

5. Clerical Error

A clerical error is human failure which is made during the transformation of any physical representation (such as a coding sheet) of the program or parts of the program to another physical representation (such as punch cards).

6. Debugging Error

Inappropriate use of debugging tools, insufficient or inappropriate selection of test cases, test data or

misinterpretation of debugging results.

7. Testing Error

Inappropriate or insufficient test cases or test data, misunderstanding of functional requirements of the program, deviation from test plan or misinterpretation of test results.

8. Implementation Error

Unexpected environmental problem which occurs during the implementation of a program which could not have been anticipated in a previous stage of the software development. This includes cases such as changes of hardware by the manufacturer and performance problems with operating systems.

According to the definition of software errors, an error which does not lead to an incorrect action or result within the program cannot be identified as a software error. For example insufficient test cases during debugging of a subroutine may not lead to a software error. However, the potential exists for undetected software errors when test cases are insufficient. According to the definition of debugging errors, errors of this category do not necessarily affect the program. Therefore debugging errors will not be detected very often; however, because of being a potential

error source, errors of this category were considered to be important. During the experiment described in chapter 4 all debugging errors were recorded. Similarly design errors, which were detected during a design review, or faulty comments were recorded. This was of great importance, since it yields significant information about error sources.

B. DEFINITION OF ERROR TYPES

The list of error type definitions given below identifies the most common types of errors within the major categories of software errors. If appropriate, examples are given for clarity.

These definitions are important because it has been discovered that without a definition of error types, errors of the same type could be recorded in different ways, not only by different programmers, but also by the same person at different times.

1. Design Errors

The following types of errors apply to both categories "System Design Errors" and "Program Design Errors":

Di : Communication Error

An error occurs due to improper communication

between members of a programming team or due to inappropriate systems design specifications. Typical cases are conflicting interpretation of common data and misunderstanding of module functions.

D2 : Design Negligence

Parts of program or system design specifcations are neglected or necessary cooperation with the originator(s) of the specification have been omitted, such as neglecting desired output formats or lack of documentation.

D3 : Forgotten Cases or Steps

While solving the problem one or more cases have not been considered, such as neglecting leap years for calculation of dates or necessary step(s) to solve the problem have been omitted, such as determining the status of peripheral devices before opening I/O, or forgotten conversion from decimal to binary or vice versa.

D4 : Timing Problems

Misconception in scheduling program events, such as assigning inappropriate priorities or changing data before completion of I/O.

D5 : Errors in I/O Concepts

Errors made in the design of input or output, such as misunderstanding of hardware requirements of I/O devices or channels. Typical cases are

misunderstanding of the capabilities of terminals or graphics devices, neglecting the size of lines on a lineprinter or the interrupt capabilities of peripheral devices.

D6 : Data Design Error

The data design does not fit the needs of the program. Typical examples are wrong sizes of arrays, buffers or freelists. Also cases of inappropriate data definitions, such that the chosen data types do not accomplish the desired precision. This error type also includes design of inappropriate I/O formats and improper design of common data.

D7 : Initialzation Error

Wrong or incomplete initialization such as forgotten initialization of global pointers, arrays or flag variables or forgotten initialization of interrupt handlers.

D8 : Inadequate Checking

Checking of variables or input data is incomplete or wrong according to program specifications.

D9 : Extreme Conditions Neglected

Data or machine dependent extreme conditions are neglected, such as numeric values which cause underflow, overflow, excess of array limits or maximum values of real numbers or integers, and occurrence of zero or negative values due to round off.

D10: Sequencing Error

Program events or decisions scheduled in wrong sequence such as transformation from problem statement into a sequence of operations.

D11: Indexing Error

Faulty index calculation such as designing an erroneous algorithm for accessing array elements or items of a table.

D12: Loop Control Errors

Error in controlling either an iterative or conditional repetitive algorithm, such as basing exit or break decisions on conditions which can never occur.

D13: Misuse of Boolean Expression

An error is made in using Boolean operators, such as constructing a false inverse of an expression. A trivial example of this error type is solving

$$(x + y)' * z = (x' + y') * z$$

instead of:

$$(x + y)' * z = (x' * y') * z$$

where +, *, ' are the Boolean operators "OR", "AND", and "NOT" respectively.

D14: Mathematical Error

The mathematical solution of a problem within the programming project does not or not always represent the correct result according to the program specifications. This includes cases in which a

mathematical equation is simply wrong, like a wrong formula for calculating a mean or variance of given set of numbers and cases in which the mathematical solution does not apply for all possible inputs, such as solving a square root for negative expressions.

D15: Representation Error

An error is made in the process of the physical representation of thoughts, such as writing design documentation different from what the designer had intended.

D16: Misunderstanding of Problem Specifications

Error resulting from misunderstanding the problem specifications. For example a decision table, developed during design, includes some undesired combinations of actions.

D17: Other Design Errors

Any other design error which is not one of the types listed above.

2. Coding Errors

C1 : Misunderstanding of Design

An error made in interpreting the design specifications, such as misunderstanding an algorithm, data

descriptions or critical terms, such as mathematical expressions which leads to an error in implementation of the underlying design.

C2 : Negligence

Parts of the design are neglected or necessary cooperation with the designer(s) is omitted. This will include cases where programmers are uncertain about the design or parts of it and fail to take appropriate action, such as requesting an explanation from the designer(s) or obtaining the missing information from other sources such as project specifications, hardware or software manuals.

C3 : I/O Format Error

Use of improper I/O formats in coding, such as using formatted I/O when nonformatted I/O is specified or using inappropriate record lengths.

C4 : Misplaced Data Declaration

Error made in declaring data either in the wrong block level (while coding in a block structured language) or in an inappropriate position within the source code.

C5 : Multiple Data Declaration

Multiple declarations of same data.

C6 : Missing Data Declaration

Necessary data declaration left out.

C7 : Inadequate Data

Error made in choosing size or type of data, such as declaring single precision instead of double precision.

C8 : Initialization Error

Wrong or forgotten initialization.

C9 : Error in Parameter Passing

Passing of parameters is incomplete, wrong or types do not match.

C10: Inadequate or Forgotten Checking

Checking of input is either incomplete or wrong according to program design specifications, such as not checking for invalid input within a subroutine which receives characters from a user terminal. A typical case of this error type occurs when numerical input is expected and is converted to binary without checking the validity of the input character. Another case is an inadequate check of the size of the input number.

C11: Level Problems

While coding in a block structured language an error results from confusing block levels, such as misunderstanding the scope of variables, e.g. an ALGOL programmer tries to use variables or data of a lower level within the main level (level 1) of a program.

C12: Missing Declarations of Block Limits

While coding in a block structured language mandatory declarations of block level limits such as BEGIN, END are left out.

C13: Case selection error

An error is made in connection with a CASE statement or computed GO TO statement or an equivalent construct. For example in an implementation of a GO TO SWITCH in CMS-2 an error occurs due to wrong sequence of labels. CMS-2 is a programming language used for tactical programming within military applications [36].

C14: GO TO Problems

An error is made in connection with the use of a GO TO statement.

C15: Comment Error

Stating a faulty or misleading comment.

C16: Forgotten Delimiter

Delimiter left out.

C17: Inconsistency in Naming

An error is made when naming an identifier in different ways. For example a subroutine declared as GETCHAR is called by using GETCH.

C18: Wrong Use of Nested IF Statements

An error is made during the construction of a nested IF statement. Errors of this type occur in transforming a flowchart or decision table logic

into nested IF statements or using an incorrect format which will cause a compile error or associating any of the ELSE cases with a wrong IF clause.

C19: Indexing Error

Indexing is not appropriate or left out.

C20: Inconsistent Use of Variables or Data

An error occurs because variables or data are used for conflicting purposes. For example usage of a temporary storage location in different parts of the program could be implemented in such a manner that information is overwritten by some subroutine but still be needed by another subroutine. Another example is the usage of common variables for different purposes like pointer and index.

C21: Sequencing Error

An improper sequence of statements which leads to an error. For example an exchange of array elements is programmed

```
TEMP ← A(i)
A(j) ← TEMP
A(i) ← A(j)
```

which does not accomplish the desired exchange of the two elements.

C22: Flag Usage Problems

An error is made in using flags in order to control program logic.

C23: Syntax Error

Violation of syntax rules of the programming language. This includes errors in connection with transformation of boolean expressions and arithmetic expressions.

C24: Loop Control Error

Error in implementing the control structures of an iterative or conditional repetitive algorithm.

C25: Incorrect Exit from Subroutines

An error occurs in the implementation of an exit from a subroutine.

C26: Language Usage Problems

An error is made due to idiosyncrasies of the programming language which are not thoroughly understood by the programmer.

C27: Forgotten Statements

An error is made by leaving out a necessary statement which the programmer knew was necessary.

C28: Representation Error

A coding error which is introduced during the process of the physical representation of thoughts, such as writing a statement different from what one intended.

C29: Control Sequence Error

An error is made which causes the program to follow a wrong branch after a decision.

C30: Incorrect Subroutine Usage

Using the wrong subroutine or using a subroutine inappropriately, e.g. a programmer's expectation of what a subroutine will accomplish is different from what the subroutine actually does.

C31: Other Coding Errors

Any other coding error which is is not one of the types listed above.

3. Clerical Errors

A1 : Manual Error

An error resulting from lack of motoric skill or temporary manual misfunction, such as

- Errors of Commission (e.g. writing or typing "busu" instead of "busy")
- Errors of Omission (e.g. writing or typing "BOWN" instead of "BROWN")
- Errors of Transposition (e.g. writing or typing "hte" instead of "the")

A2 : Mental Error

An error resulting from a mental misfunction, such as

- Perceptual Errors (e.g. reading "0" instead of "O", 11 instead of ll or "DOG" instead of "DOD")

- Expectation Errors (e.g. in a given context a certain word or expression is expected and the actual word or expression is overlooked, such as assuming "FOR I:=N STEP 1 UNTIL M DO" where "FOR N:=1 STEP -1 UNTIL M DO" is the the actual implementation of an iterative loop in ALGOL W.)

A3 : Procedural Errors

Given a proper assignment for a clerical task which implies some well defined sequence of actions which could be expected from the assignment, some of the steps are either forgotten or not carried out properly, such as some lines of code on a coding sheet are omitted by the keypunch operator or card correction procedures are not followed appropriately or inserted cards are misplaced.

A4 : Other Clerical Errors

Any other clerical error which is not one of the types listed above.

4. Debugging Errors

B1 : Inappropriate Use of Debugging Tools

Wrong selection of debugging tools, such as traces, snapshots or dumps or using them inappropriately. For example using a snapshot before instead of after

a certain instruction or taking a dump of wrong memory locations.

B2 : Insufficient or Inappropriate Selection of Test Cases or Test Data

The test cases or test data provide insufficient test coverage of the program or fail to provide testing of critical parts of the program.

B3 : Misinterpretation of Debugging Results

An inappropriate action is taken due to misinterpretation of debugging results, such as a wrong result is overlooked or misunderstanding of a correct result leads to additional errors.

B4 : Misinterpretation of the Error Source

An inappropriate action in correcting a discovered software error that results from wrong assumptions about the error source, such as correcting the symptoms of an error rather than its cause or the programmer fails to foresee the impact of a change on other parts of the program and inserts additional errors into the program.

B5 : Negligence

Errors resulting from negligence in debugging. For example the implementation of an error correction is assumed to be error free without further checking.

B6 : Other Debugging Errors

Any other debugging error which is not one of the

types listed above.

5. Testing Errors

T1 : Inadequate Test Case(s) or Test Data
Selected test case(s) or test data do not properly test the program.

T2 : Misinterpretation of Test Results
An inappropriate action is taken due to misunderstanding of test results, such as a wrong result is overlooked or misunderstanding of a correct result leads to an implementation of additional errors.

T3 : Misinterpretation of Problem Specifications
Error(s) resulting from misunderstanding the program specifications, such as incorrect interpretation of user defined functions which leads to an insertion of an error or to over looking an existing error.

T4 : Negligence
Errors resulting from negligence in testing. For example a tactical real time system is merely tested within specified limits and tests fail to examine the behavior of the system for cases where limits are exceded. In this particular example negligence can cause serious results in critical situations.

T5 : Other Testing Errors

Any other testing error which is not one of the types listed above.

In order to get meaningful results, the above definitions were used throughout the project. This list is not considered complete. However, it permitted identification of software error types for the purpose of this experiment. Some of the listed error types could be further subdivided into subsets of each type if this were desired.

To contribute to analysis of error sources, it was decided to list the same error type which occurs in different software development phases under different categories. For example one type of error which has the same name (loop control error) may occur in both design and coding phases. However, the cause of the error could be different.

Error severity is related to the effect of the error on system operation. There is no direct relationship between error type and severity. For example mistyping of a single character of a variable name in FORTRAN can cause strange results, whereas the same error made in a comment may not affect the program at all. The consideration of error severity was beyond the scope of this project.

IV. EXPERIMENT DESCRIPTION

A. GENERAL REMARKS

Error data were gathered in four programming projects ranging from small to large. During 280 man hours of total project time more than 2000 source statements were produced. A total of 173 errors were recorded.

The scope of the programming projects was limited to the following software development phases:

- Design and Design Review
- Coding
- Debugging
- Testing

B. DESIGN OF THE EXPERIMENT

The experiment was designed to guarantee the most accurate recording of relevant error information. Therefore, for each of these phases, an appropriate form was designed to give a firm guideline to the experiment programmer for recording the entire development of a program including necessary details about errors. The error data which were recorded on these forms for project 1, 2, 3 and 4 are given in Appendices A, B, C and D respectively. For each project

the recorded data was analyzed with respect to the following factors:

- Number of source statements produced
- Man hours spent on the project
- Man hours spent in each of the software development phases
- CPU time used for compiles
- CPU time used for test and debug runs
- Number of test and debugging steps needed
- Project time used to correct errors
- Project time between error detections
- Occurrence of errors with respect to software development phases
- Detection of errors with respect to software development phases
- Time history of error detection and correction
- Complexity measures of subroutines

A summary of this analysis is documented at the end of each appendix.

Since it would be too difficult to obtain error data in a commercial environment, it was neccessary to obtain the data by using a carefully controlled experiment in an academic environment. The considerable overhead necessary to record all relevant data was not included in project time. It probably would not be feasible to conduct this kind of experiment in a commercial environment because:

- The overhead would be very costly. It is unlikely that management would support an experiment of this type.
- Competition among programmers would probably not allow accurate results to be obtained.
- Programmers may feel that the error data would be used for performance evaluation.

In addition to gathering quantitative error data an attempt was made to obtain other information about the nature of errors such as "why was the error made?" or "how was the error discovered?". This information could help to devise methods for avoiding certain types of errors. For each subroutine a directed graph representation was analyzed with respect to the following complexity measures:

- Number of Nodes
- Number of Arcs
- Number of Statements
- Number of Paths
- Reachability of Nodes
- Cyclomatic Number [32]

The results of this analysis are shown together with the directed graph representation of the programs in the appendices. For some structures these measures are not shown because the number of paths and the reachability index was too large to calculate.

C. PROGRAMMING ENVIRONMENT

The underlying program specifications of all projects were well defined and not known by the experiment programmer before the experiment. The experiment programmer was familiar with the objectives of the experiment and willing to record accurate data to the best of his knowledge, even if recorded facts seemed to be unfavorable for himself.

Throughout the experiment software development was done in a structured and phase oriented approach using structured programming and known software development techniques, such as top-down design, modularization, decision tables, etc..

As a programming environment the OS/360 system was chosen as one of the most representative batch processing systems available. Batch processing was always used except for the final test and debug runs for project # 4, because this program was designed to run in a time sharing environment. During the experiment all environmental factors, such as operating system and equipment being used and the programming language remained unchanged. Factors related to the experiment programmer, such as ability, knowledge, and experience did not vary considerably during the experiment.

V. QUANTITATIVE RESULTS

A. DISTRIBUTION OF PROGRAMMING EFFORT

In Table 1 the distribution of project time is shown with respect to the major software development phases.

PROJECT PHASE	PROGRAMMING EFFORT (MAN HOURS) Project # 1	Project # 2	Project # 3	Project # 4	Total
DESIGN	5.0 (22.9%)	31.0 (24.8%)	7.0 (21.2%)	24.0 (23.8%)	67.0 (23.9%)
CODING	7.0 (32.1%)	26.0 (20.8%)	4.0 (12.1%)	24.5 (24.3%)	61.5 (21.9%)
DEBUGGING	4.0 (18.3%)	55.0 (44.0%)	3.0 (9.1%)	41.5 (41.1%)	103.5 (36.9%)
TESTING	5.8 (26.6%)	13.0 (10.4%)	19.0 (57.6%)	11.0 (10.9%)	48.8 (17.4%)
SUM	21.8	125.0	33.0	101.0	280.8

TABLE 1

DISTRIBUTION OF PROJECT TIME DURING THE EXPERIMENT

B. ORIGIN OF ERRORS VS DETECTION OF ERRORS

The final statistics of each project contains tables showing the numbers of errors made and the number of errors

found with respect to each of the software development phases. A summary of these tables is presented in Table 2.

PROJECT PHASE	# OF ERRORS FOUND Project # 1	Project # 2	Project # 3	Project # 4	TOTAL (Percentage)
DESIGN	2	1			3 (1.7 %)
CODING	22	21	1	4	48 (27.7 %)
DEBUGGING	19	53	3	45	120 (69.4 %)
TESTING	1			1	2 (1.2 %)
SUM	44	75	4	50	173 (100 %)

PROJECT PHASE	# OF ERRORS MADE Project # 1	Project # 2	Project # 3	Project # 4	TOTAL (Percentage)
DESIGN	5	20		10	35 (20.2 %)
CODING	38	53	4	37	132 (76.3 %)
DEBUGGING	1	2		3	6 (3.5 %)
TESTING					0 (0.0 %)
SUM	44	75	4	50	173 (100 %)

TABLE 2

ERROR DETECTION VS ERROR ORIGIN

C. ERROR CORRECTION TIME AND TIME BETWEEN ERRORS

1. Error Correction Time

The mean time to correct an error calculated over all 173 errors was 12.2 man minutes.

2. Time Between Error Detections

The mean time between error detections calculated over all projects was 61.6 man minutes.

D. ERROR TYPES

All errors which occurred during the experiment could be identified by one of the previously defined error types. An overview of error types with respect to their frequency of occurrence is presented in Table 3. Error types which did not occur are not listed. In addition to the information given in Table 3 a bar chart (Table 4) is shown which identifies the most common error types.

ERROR TYPE	NUMBER OF ERRORS Project # 1	Project # 2	Project # 3	Project # 4	TOTAL
D3	3	3		2	8
D7				1	1
D9	1	10			11
D10		1			1
D11	1	3			4
D12		2		5	7
D13				1	1
D15		1		1	2
C1	1	1			2
C4	1				1
C5	1				1
C6	1		1	2	4
C7				1	1
C8	2	1			3
C9		2		1	3
C10	1	2			3
C11	3	2			5
C12	4	2		1	7
C15	1				1
C16		1			1
C17	1	5		3	9
C19	1		1		2
C20	2	1			3
C21		2	1	2	5
C23	6	5		1	12
C24	2	1	1		4
C26		1			1
C27	2	3		4	9
C28	5	6		6	17
C29		2			2
C30		1			1
A1	2	14		16	32
A2	2	1		1	4
A3				1	1
B3	1				1
B4		2		1	3

TABLE 3

ERROR DISCOVERY WITH RESPECT TO ERROR TYPES

```
ERRUR  ¦
TYPE   ¦  FREQUENCY OF OCCURRENCE
-------+-------------------------------------------------------------
D3     ¦********
D9     ¦***********
D11    ¦****
D12    ¦*******
C6     ¦****
C11    ¦*****
C12    ¦*******
C17    ¦*********
C21    ¦*****
C23    ¦************
C24    ¦****
C27    ¦*********
C28    ¦*****************
A1     ¦********************************
A2     ¦****
Other  ¦
Errors ¦***********************************
-------+-------------------------------------------------------------
```

TABLE 4

MOST FREQUENT ERROR TYPES

E. ERROR OCCURRENCE RELATED TO COMPLEXITY MEASURES

One of the objectives of the experiment was to determine the relationship, if any, between error occurrence and the structural properties of subroutines. Of particular interest was the analysis of structural properties with respect to the error simulation work done by Schneidewind, Howard and Kirchgaessner [33]. In addition the cyclomatic number as defined by McCabe [32] for software engineering purposes, was used in the analysis. The results of this analysis are shown in Table 5. The following definitions

and terms are used in Table 5:

Np Number of Paths (minimum number of paths: no loop traversed more than once in succession)

V Cyclomatic Number: number of independent circuits = number of arcs - number of nodes + 2

R Reachability: summation, over the nodes, of number of ways of reaching a node

r Average reachability: R/number of nodes

S Number of Source Statements

e Number of Errors Found in Actual Program

Tf Labor Time Required to Find Errors (Since Previous Error Detection)

Tc Labor Time Required to Correct Error

All of the above are with respect to a single program. The definition of V includes an implicit arc connecting the start and terminal nodes (strongly connected graph).

TABLE 5

COMPLEXITY MEASURES VS. ERROR PROPERTIES

Part A - Procedures With One or More Errors

Project/ Procedure	COMPLEXITY MEASURES No	V	R	r	S	e	ERROR PROPERTIES $\sum T_f$ (Man-mins)	$\sum T_c$ (Man-mins)
1/1	2	2	7	1.4	14	1	35	10
1/5	*	6	*	*	26	5	95	53
1/6	*	5	*	*	7	2	37	35
1/8	*	5	*	*	21	1	35	15
1/9	2	2	8	1.333	6	1	115	20
2/1.19	1	1	2	1.0	3	1	10	10
2/1.23	1	1	2	1.0	11	1	110	10
2/2.2	2	1	4	1.0	8	1	15	10
2/7	3	2	7	1.4	15	3	230	45
2/9	72	8	370	19.474	45	3	140	185
2/10	9	4	25	2.778	18	1	10	5
2/11	*	6	*	*	54	3	950	65
2/12	5	2	13	1.444	34	2	300	30
2/15	*	4	*	*	19	1	5	1
2/16	*	5	*	*	30	2	150	20
2/18	12	4	50	2.941	26	1	5	15
2/21	*	16	*	*	94	8	750	145
3/3	2	2	6	1.5	13	1	60	5

* Very large value

TABLE 5
(continued)

Part A - Procedures With One or More Errors

Project/ Procedure	No	V	R	r	S	e	$\sum$Tf (Man-mins)	$\sum$Tc (Man-mins)
4/7	40	6	151	3.438	83	1	120	5
4/13	16	5	64	4.267	28	1	20	15
4/14	*	8	*	*	37	5	255	65
4/15	4	3	12	2.0	13	2	40	35
4/21.3	7	3	34	4.857	16	1	0	5
4/22	*	7	*	*	34	1	160	30
4/23	18	5	60	4.615	24	1	125	10
4/27	5	4	23	2.091	18	3	360	120
4/28	*	5	*	*	35	2	90	50
4/29	321	13	1468	54.370	49	5	1125	160
4/30	6	4	24	2.4	19	1	60	30
4/31	*	4	*	*	27	1	30	10
4/33	14	4	76	7.6	17	2	135	10

* Very large value

TABLE 5
(continued)

Part B - Complexity Measures for Procedures
With Zero Errors

Project/ Procedure	Np	V	R	r	S
1/2	3	2	8	1.333	6
1/3.1	1	1	2	1.0	8
1/3.2	1	1	2	1.0	11
1/3.3	1	1	2	1.0	4
1/4	6	3	26	4.333	18
1/7	7	3	40	5.0	15
2/1.1	1	1	2	1.0	3
2/1.2	1	1	2	1.0	3
2/1.3	1	1	2	1.0	3
2/1.4	1	1	2	1.0	3
2/1.5	1	1	2	1.0	3
2/1.6	1	1	2	1.0	3
2/1.7	1	1	2	1.0	3
2/1.8	1	1	2	1.0	3
2/1.9	1	1	2	1.0	5
2/1.10	1	1	2	1.0	5
2/1.11	1	1	2	1.0	5
2/1.12	1	1	2	1.0	13
2/1.13	1	1	2	1.0	3
2/1.14	1	1	2	1.0	3
2/1.15	1	1	2	1.0	3
2/1.16	1	1	2	1.0	3
2/1.17	1	1	2	1.0	3
2/1.18	1	1	2	1.0	3
2/1.20	1	1	2	1.0	3
4/1.1	1	1	2	1.0	2
4/1.2	1	1	2	1.0	2
4/1.3	1	1	2	1.0	7
4/1.4	1	1	2	1.0	5
4/1.5	1	1	2	1.0	7
4/1.6	1	1	2	1.0	5
4/1.7	1	1	2	1.0	5
4/1.8	1	1	2	1.0	5
4/1.9	1	1	2	1.0	5
4/1.10	1	1	2	1.0	4
4/1.11	1	1	2	1.0	3
4/1.12	1	1	2	1.0	3
4/1.13	1	1	2	1.0	3
4/1.14	1	1	2	1.0	3
4/1.15	1	1	2	1.0	3
4/1.16	1	1	2	1.0	3
4/1.17	1	1	2	1.0	3

TABLE 5
(continued)

Part B - Complexity Measures for Procedures With Zero Errors

Project/ Procedure	Np	V	R	r	S
4/1.18	1	1	2	1.0	3
4/1.19	1	1	2	1.0	5
4/1.20	1	1	2	1.0	5
4/1.21	1	1	2	1.0	6
4/1.22	1	1	2	1.0	9
4/1.23	1	1	2	1.0	6
4/2.1	2	1	4	1.0	8
4/2.2	2	1	4	1.0	9
4/2.3	2	1	4	1.0	9
4/3	2	2	6	1.5	4
4/4.1	2	2	8	1.6	7
4/4.2	2	2	8	1.6	9
4/5	8	4	38	4.222	56
4/6	4	1	6	1.0	24
4/8.1	2	2	8	1.6	13
4/8.2	2	2	8	1.6	13
4/8.3	2	2	8	1.6	10
4/8.4	2	2	8	1.6	9
4/8.5	2	2	8	1.6	12
4/9	16	5	95	7.917	21
4/10	12	5	65	4.643	49
4/11	7	3	38	5.429	19
4/12	*	4	*	*	20
4/16.1	2	2	6	1.5	6
4/16.2	2	2	6	1.5	12
4/16.3	2	2	6	1.5	9
4/16.4	2	2	6	1.5	10
4/17	16	1	18	1.0	21
4/18	8	4	42	3.818	21
4/19	4	3	16	2.667	11
4/20	3	2	9	1.125	13
4/21.1	7	3	34	4.857	14
4/24	16	7	83	4.368	19
4/25.1	2	2	8	1.333	15
4/25.2	2	2	8	1.333	10
4/25.3	2	2	8	1.333	17
4/26	3	3	15	1.5	19
4/32	7	3	36	5.143	15
4/34	2	2	5	1.25	15

* Very large value

Using the results of Part A in Table 5, the following correlation coefficients were calculated:

V vs.	e :	.7834
S vs.	e :	.5880
V vs.	ΣTf :	.6734
V vs.	ΣTc :	.7229
S vs.	ΣTf :	.5902
S vs.	ΣTc :	.5091
V vs.	S :	.7903

The relationship between mean complexity values and error occurrence is shown in Table 6.

	Zero Errors 81 Procedures	One Or More Errors 31 Procedures
Mean Cyclomatic Number (V)	1.68	4.74
Mean Number of Source Statements (S)	9.33	27.23

TABLE 6

RELATIONSHIP BETWEEN MEAN COMPLEXITY VALUES AND ERRORS FOUND

VI. SUMMARY AND CONCLUSIONS

A. GENERAL REMARKS

Due to limited man power and time constraints the amount of test data was too small to allow major conclusions. Furthermore the scope of the experiment did not include some important programming problems such as real time and mathematical problems. However, taking all constraints into consideration some qualitative aspects were recognized which could be important factors in subsequent studies.

B. SIGNIFICANCE OF ERROR TYPES

Although the categorization and definition of error types was not considered complete, all errors detected in the experiment could be identified using these error definitions. The most frequent error type "A1" (Manual Error) seems to be related to the number of source statements whereas the frequencies of the remaining errors seem to be related to the complexity of the programming problems.

The recording of errors with respect to their types contributes to a learning process which enables the programmer to reduce the number of errors made on subsequent projects. It was felt that this concept could also be effectively used in a commercial software production environment. The error recording would have to be private to the individual

programmer and used by the programmer alone. Otherwise it would have a negative influence on working atmosphere and performance. If the evaluation process were assisted by appropriate software packages, the overhead in analyzing errors would be reduced to a great extent. The overhead of recording the errors for the individual programmer tends to decrease after the recording system has been learned.

C. COMPLEXITY MEASURES

Several measures of complexity calculated for each subroutine are presented in the appendencis together with subroutine directed graph representation. When the number of errors found in procedures was correlated with cyclomatic number and number of source statements, the correlation coefficients were higher than for other complexity measures [34]. It also appeared that these two measures were related to the total error detection and total error correction times. It was learned that trying to keep the cyclomatic number small not only reduced the number of errors but also contributed to the reduction of debugging and testing effort. This experience supports the results of McCabe's analysis of structural properties of programs with respect to difficulties in testing and debugging [32]. It should be noted that the influence of other factors such as usage of structured programming techniques and working habits of the experiment programmer also contributed to these results.

D. SOFTWARE ENGINEERING ASPECTS

Throughout the experiment comments were made to indicate possible clues about error sources or ways to avoid certain types of errors. An evaluation of these comments showed that some of these errors would have been avoided by the usage of decision tables or applying a proper desk test. Many key punch errors of type "A1" were not recognized during the process of punching cards, because the design of the key punch machines (IBM 29) does not provide an immediate control of characters being punched. It is desirable to see the result of each key stroke instantaneously. The frequency of error occurrence of type "Manual Error" in former projects done by the same programmer using console editing was considerably lower than the error frequency discovered in this experiment.

Software engineering concepts such as top-down design, design review and Structured Walk Through were used throughout all projects. These methods had a major influence on the software development process and were considered to be extremely useful.

The usage of ALGOL W had a major influence on the programming style. Variable names were chosen to make programs self-documenting to a great extent. Therefore most program parts were easy to read and to understand which implied a reduction of the debugging effort.

Another concept which was successfully employed was the idea of using identifiers which differ by two or more characters. The idea of implementing some kind of "Hamming Distance" [35] between identifiers was derived from bitter experience in former projects and helped to avoid errors which occur when a single error of type "Manual Error" or "Mental Error" produces a different identifier which matches with one of the previously defined identifiers. Thus this kind of error did not occur during the compiling process.

E. CONCLUSIONS

Errors in software projects appear to be related to program structure and to the number of source statements. Although this experiment was not large enough to state these relations in a formal manner, the results were encouraging. Cyclomatic number and number of source statements could prove useful as guidelines for managing programming projects.

The usage of top-down design, structured programming techniques and other software development tools should be mandatory for programming projects. The recording and evaluation of software errors would be a reasonable approach to avoid errors in programming projects and to reduce cost.

Detailed debugging on a modular basis and development of large programs by stepwise integration of thoroughly debugged modules seems to be a good strategy to achieve

reliable programs.

F. RECOMMENDATIONS

Similar experiments would reveal more of the relations between software errors and complexity of programs. More definite information about the utility of complexity measures could be obtained by using these measures on large-scale software projects in a commercial production environment.

APPENDIX A

PROJECT DESCRIPTION

Project # : 1

Project title : PALINDROMES

Programmer : HOFFMANN

Programming Language : ALGOL

Programming environment: IBM/360/67,OS/MVT,BATCH

Design notes : see ANNEX A

Program listing : see ANNEX B

Coding notes : see ANNEX C

Debugging notes : see ANNEX D

Error Listing : see ANNEX E

Final statistics : see ANNEX F

Graphical representation : see ANNEX G

Test phase description: see ANNEX H

Starting date: 28 JAN 77   Ending date: 6 FEB 77

EXPERIMENT DESCRIPTION

1. Project description:

FIND PALINDROMS

A palindrome is defined as a character string of length n, which has the following characteristics:

character[i] = character[n - i - 1] for all i from i=0 to i=n/2 (integer division) and n >= 2

Given an input string of length n, where 2 <= n <= 256, find all occurrences of palindromes.

The input is given in form of punched cards.

The length of the string ("l") preceeds the input of the character string.

All information being read should be printed.

Give as a result the number of palindromes of each length and the position of the starting character of each palindrome in card number and character position.

Find only maximal length palindromes. (i.e. "ABC CBA" should be recorded as the palindrome of maximal length, whereas the included palindromes "BC CB" and "C C" should be excluded.)

2. Programmer's background:

a) Experience in programming:

Oct 1970 - May 1971	Programming courses
May 1971 - April 1972	Module Programmer
May 1972 - June 1974	work in Test and Simulation Department at the NAVAL COMMAND AND CONTROL SYSTEMS COMMAND (FEDERAL GERMAN NAVY) Testing of tactical real time systems

March 1975 - Jan 1977   Student at the NAVAL POSTGRADUATE SCHOOL, Monterey, Computer Science

b) Experience in testing:

Two years of work in testing and simulation.

c) Experience in the area of the given problem: None.

d) Experience in the progamming language being used:

Experience over a period of 18 months in more than 10 programming projects. (Total number of source statements produced during that time was more 4000.)

3. Psychological factors:

a) Did the programmer like the project? Yes.

b) How does the programmer like the programming language?

Favorite programming language.

c) Was the programmer satisfied by the way the problem was specified?

Only minor criticism.

d) How did the programmer like the programming environment?

The facilities (study room, card punch room) were not conducive to efficient programming because of restricted

space, bad lighting and noise.

e) Other factors:

The recording of the experiment's data during the project affected speed and concentration considerably.

4. Comments on Documentation

For the documentation of each software development phase a special documentation form has been developed. These forms are designed to provide a firo guideline for the experiment programmer to record all data of interest for subsequent error analysis.

- Begin and end of each step was recorded with respect to day and time.

- Each error was recorded when it is discovered. The error was then identified by a unique error number (1,2,...). Furthermore the time of discovery and the error type (using error types listed in ANNEX F) were recorded.

- If appropriate, comments about error discovery, reason why the error was made, etc. were documented in ANNEX E2.

- For each error the phase in which the error was made, the phase in which the error was discovered and the time spent to correct the error was recorded in ANNEX E1.

- For each step in any one of the software development phases the day/time of begin and end was recorded. In addition, the time (in man hours) for each step was recorded. This excludes the overhead used for documentation of the experiment data.

WORKSHEET FOR DESIGN PHASE AND DESIGN REVIEW PHASE OF PROJECT # 1

STEP #	PROBLEM AND PLANNED SOLUTION	ALTERNATE SOLUTIONS	DAY TIME	MAN HOURS /STEP	ERROR #	COMMENT
1	Top-Down Design: Use the following building blocks: -Initialization -Read and write Input Cards -Find all Palindromes which are not totally included in a larger palindrome Find also palindromes which are overlapping: ABBAXA will be recorded as two palindromes: ABBA and AXA -Write all palindromes being found -The necessary inputs 1. length of string 2. character string of given length are expected from cards.	-Do not allow overlapping palindromes. This allows two solutions: 1. Always record the first palindrome. 2. Always record the larger palindrome. If palindromes are of equal length, then record the first one.	1/28 1040 1/28 1330	1.5		
2	Specify Initialization: -Initialize all arrays and variables -Write explanatory text -Read length of input string and check(assert 2<= length of string <= 256) -Set integer field size to 5 -Use all 80 columns of the input cards (specify as compile parameter CARDLIMIT)	Cardlimit could be specified by user via input parameter.	1/28 1330 1/28 1400	.5		

Remarks:

Because of the small size of the project the system design phase is omitted.

Total man hours spent in design: 5.0

WORKSHEET FOR DESIGN PHASE AND DESIGN REVIEW PHASE OF PROJECT # 1

STEP #	PROBLEM AND PLANNED SOLUTION	ALTERNATE SOLUTIONS	DAY TIME	MAN HOURS /STEP	ERROR #	COMMENT
3	Specify Reading and Writing of Input Cards:		1/28 1400			
	-Number of input cards is dependent on CARDLIMIT and specified length of character string. It is calculated: Number of input cards= (length of string -1)/cardlimit + 1			.5		
	-Maintain a card counter					
	-Read and write each card preceeded by the appropriate card number.		1/28 1500			
4	Data Design:		1/28 1500			
	-Use a text buffer (Text) (String Array Indexing from 1 through 256)		1505		1	will be redesigned during design review(D3)
	-Use an I/O buffer(Cardbuffer)					
	-Use two integer arrays to record begin and end positions of all palindromes. (Begin-of-palidrome,End-of-palindrome)			.5		
	-Use other self documenting common data: (Cardlimit, Length-of-text, Bufferposition, Cardcounter, Palindromecounter)					
	-Use Index variables IX,JX		1/28			
	-Use Local variables as needed		1530			

Remarks:
Error #1 was made during step 2.

WORKSHEET FOR DESIGN PHASE AND DESIGN REVIEW PHASE OF PROJECT # 1

STEP #	PROBLEM AND PLANNED SOLUTION	ALTERNATE SOLUTIONS	DAY TIME	MAN HOURS /STEP	ERROR #	COMMENT
5	Find Palindromes: -Scan text from left ro right. If palindromes of minimal length (i.e. length is 2 or 3) are found, call a subroutine which checks whether or not the palindrome is the center of a larger palindrome.	-Scan text from left to right. Consider every character as the possible beginning of a palindrome. Check for all possible lengths going backward from the end of the text string down to a string length of 2.	1/28 1530 1/28 1550	.3		
6	Recording of Palindromes: -Record a palindrome only if it is not entirely included in any of the previously recorded palindromes. -It is also necessary to check whether any of the previously recorded palindromes is entirely included in the palindrome last detected. -A flag should be set to make sure that only maximum length palindromes will be printed.		1/28 1550 1/28 1620	.5		
7	Write all Palindromes: -Identify the position of each palindrome by begin and end with respect to input card number and position on the card. -Using the position in the text string and the parameter CARDLIMIT the input card number and the desired character positions can be calculated.		1/28 1620 1/28 1630	.2		

Remarks:

WORKSHEET FOR DESIGN PHASE AND DESIGN REVIEW PHASE OF PROJECT # 1

STEP #	PROBLEM AND PLANNED SOLUTION	ALTERNATE SOLUTIONS	DAY TIME	MAN HOURS /STEP	ERROR #	COMMENT
8	Design Review:		1/28			
	-Input cards will be stored into TEXT(string array) considering CARDLIMIT.		2100			Forgotten action in step 2
	-Preceeding the writing of the recorded palindromes a proper headline should be printed. (TEXT2)		2115		2	Forgotten action in step 6 (D3)
	-If no palindromes have been found, write equivalent message. (TEXT3)			1.0		
	-Design of printout format.		1/28 2200			See remarks

Remarks:
Design of Printout Format:

Palindrome Number	Begin Card Number	Begin Character Position	End Card Number	End Character Position
n <palindrome>	bnc	bcp	ecn	ecp

Where n is the sequence number
bcn is the card number of palindrome begin
bcp is the character position of palindrome begin
ecn is the card number of palindrome end
ecp is the charcter position of palindrome end.

Annex B

Program Listing of Project # 1

```
BEGIN
COMMENT THIS PROGRAM FINDS PALINDROMES WITHIN A CHARACTER STRING
        OF MAXIMAL LENGTH = 256.
        MINIMUM LENGTH IS 2.
        ALL INPUT CARDS WILL BE LISTED.
        THE PROGRAM WILL PRODUCE A LIST OF ONLY THOSE PALINDROMES WHICH
        ARE NOT ENTIRELY INCLUDED IN A LARGER PALINDROME;

COMMENT DATA DECLARATIONS;

STRING(1) ARRAY TEXT(1::256);           COMMENT CONTAINS CHARACTER
                                                STRING;
STRING(80) CARDBUFFER;                  COMMENT I/O BUFFER FOR CARDS;
INTEGER ARRAY BEGIN_OF_PALINDROME, END_OF_PALINDROME(1::256);
INTEGER CARDLIMIT, LENGTH_OF_TEXT, BUFFERPOSITION, CARD_COUNTER,
        PALINDROME_COUNTER;
INTEGER IX,JX;                          COMMENT INDEX VARIABLES;


COMMENT INITIALIZATION;

   PROCEDURE INITIALIZE;
   COMMENT INITIALIZE ALL VARIABLES, READ LENGTH_OF_TEXT, WRITE TEXT1;
   BEGIN
   TEXT1;
   JX:=1;
   PALINDROME_COUNTER:=1;
   CARDLIMIT:=80;
   INTFIELDSIZE:=5;
   READ(LENGTH_OF_TEXT);
   IF ((LENGTH_OF_TEXT < 2) OR (LENGTH_OF_TEXT > 256)) THEN
      BEGIN
      WRITE("ILLEGAL INPUT:",
            " LENGTH OF INPUT STRING IS: ",LENGTH_OF_TEXT);
      ASSERT(FALSE);          COMMENT ABNORMAL EXIT;
      END;
   CARDBUFFER:=" ";
   END INITIALIZE;


COMMENT UTILITIES;

   PROCEDURE BLANK_LINES(INTEGER VALUE N);
   COMMENT WRITE N BLANK LINES;
   BEGIN
   INTEGER I;                 COMMENT LOCAL COUNTER;
   ASSERT(N>0);                         COMMENT SAFETY CHECK;
```

```
FOR I:=1 STEP 1 UNTIL N DO WRITE(" ");
END BLANK_LINES;


PROCEDURE TEXT1;
BEGIN
WRITE("FIND ALL PALINDROMES WITHIN THE FOLLOWING CHARACTER STRING:");
BLANK_LINES(2);
WRITE("CARD                    TEXT");
WRITE("NUMBER");
BLANK_LINES(1);
END TEXT1;


PROCEDURE TEXT2;
BEGIN
BLANK_LINES(2);
WRITE("THE FOLLOWING PALINDROMES HAVE BEEN DETECTED:");
BLANK_LINES(1);
WRITE("PALINDROME     BEGIN                    END");
WRITE("NUMBER       CARD  CHARACTER     CARD  CHARACTER");
WRITE("           NUMBER  POSITION    NUMBER  POSITION");
WRITE("-------------------------------------------------------------");
WRITECN("-------------------------------------------------------------");
END TEXT2;


PROCEDURE TEXT3;
BEGIN
WRITE("NO PALINDROMES FOUND. END OF RUN.");
END TEXT3;


PROCEDURE READ_AND_WRITE_INPUT_CARDS;
COMMENT READ INPUT CARDS ACCORDING TO GIVEN LENGTH_OF_TEXT;
BEGIN
INTEGER NUMBER_OF_INPUT_CARDS;
NUMBER_OF_INPUT_CARDS:=(LENGTH_OF_TEXT - 1) DIV CARDLIMIT + 1;
IX:=1;                    COMMENT RESET TEXT INDEX;
FOR CARD_COUNTER:=1 STEP 1 UNTIL NUMBER_OF_INPUT_CARDS DO
   BEGIN
   WRITE(CARD_COUNTER);
   WRITEON(" ");
   READCARD(CARDBUFFER);
   WRITEON(CARDBUFFER);
   BUFFERPOSITION:=0;     COMMENT RESET INDEX;
   WHILE ((IX<= LENGTH_OF_TEXT) AND (BUFFERPOSITION < CARDLIMIT)) DO
      BEGIN
```

```
        TEXT(IX):=CARDBUFFER(BUFFERPOSITION));
        IX:=IX+1;
        BUFFERPOSITION:=BUFFERPOSITION+1;
        END;        COMMENT DONE FOR ALL CHARACTERS ON A CARD;
     END;           COMMENT DONE FOR ALL CARDS;
  END READ_AND_WRITE_INPUT_CARDS;

  PROCEDURE WRITE_ALL_PALINDROMES;
  COMMENT LIST ALL PALINDROMES BEING FOUND;
  BEGIN
  INTEGER I,J;                  COMMENT LOCAL COUNTERS;
  TEXT2;
  FOR I:=1 STEP 1 UNTIL JX - 1 DO
     BEGIN
     IF END_OF_PALINDROME(I) ¬= 0 THEN
        BEGIN
        WRITE(PALINDROME_COUNTER);
        WRITEON(" ");
        WRITEON(((BEGIN_OF_PALINDROME(I) -1) DIV CARDLIMIT) + 1);
        WRITEON("  ");
        IF (BEGIN_OF_PALINDROME(I) REM CARDLIMIT = 0) THEN
           WRITEON(CARDLIMIT)
        ELSE
        WRITEON((BEGIN_OF_PALINDROME(I) REM CARDLIMIT));
        WRITEON("     ");
        WRITEON(((END_OF_PALINDROME(I) - 1) DIV CARDLIMIT) +1);
        WRITEON("  ");
        IF (  END_OF_PALINDROME(I) REM CARDLIMIT = 0) THEN
           WRITEON(CARDLIMIT)
        ELSE
        WRITEON((END_OF_PALINDROME(I) REM CARDLIMIT));
        WRITE("     ");
        FOR J:=BEGIN_OF_PALINDROME(I) STEP 1 UNTIL END_OF_PALINDROME(I)
        DO WRITEON(TEXT(J));
  WRITE("------------------------------------------------------------------");
  WRITEON("----------------------------------------------------------------");
        BLANK_LINES(1);
        PALINDROME_COUNTER:=PALINDROME_COUNTER+1;
        END;                    COMMENT END IF;
     END;                       COMMENT DONE FOR ALL PALINDROMES;
  END WRITE_ALL_PALINDROMES;


COMMENT SUBROUTINES;

  PROCEDURE PALINDROME_CHECK;
  COMMENT FIND ALL PALINDROMES WITHIN GIVEN TEXT STRING;
```

```
BEGIN
     COMMENT SCAN TEXT FROM LEFT TO RIGHT;
FOR IX:=2 STEP 1 UNTIL LENGTH_OF_TEXT DO
   BEGIN
   IF TEXT(IX-1) = TEXT(IX) THEN CONTINUE_CHECKING((IX-1),IX);
   IF IX ¬= 2 THEN
      IF TEXT(IX-2) = TEXT(IX) THEN CONTINUE_CHECKING((IX-2),IX);
   END;
END PALINDROME_CHECK;


PROCEDURE CONTINUE_CHECKING(INTEGER VALUE FIRST,LAST);
COMMENT GIVEN FIRST AND LAST AS POINTERS TO A PALINDROME
        OF SIZE 2 OR 3 THIS PROCEDURE CHECKS WHETHER OR NOT THIS
        PALINDROME IS INCLUDED IN A LARGER PALINDROME;
BEGIN
LOGICAL PALINDROME;
PALINDROME:=TRUE;
WHILE ((FIRST>1) AND (LAST<LENGTH_OF_TEXT) AND (PALINDROME=TRUE)) DO
   BEGIN
   IF TEXT(FIRST-1) = TEXT(LAST+1) THEN
      BEGIN
      COMMENT LARGER PALINDROME FOUND;
      FIRST:=FIRST-1;
      LAST:=LAST+1;
      END
   ELSE
      BEGIN
      PALINDROME:=FALSE;  COMMENT LARGEST PALINDROME FOUND;
      END;
   END;
RECORD_PALINDROME(FIRST,LAST);
END CONTINUE_CHECKING;


PROCEDURE RECORD_PALINDROME(INTEGER VALUE FIRST,LAST);
COMMENT RECORD ONLY MAX. LENGTH PALINDROMES. FLAG PREVIOUSLY
        RECORDED PALINDROMES IF THEY INCLUDED IN THE PALINDROME
        SPECIFIED BY FIRST AND LAST.
        JX IS INITIALIZED WITH 1. AFTER COMPLETION JX POINTS TO THE
        NEXT ENTRY IN BEGIN_OF_PALINDROME AND END_OF_PALINDROME;
BEGIN
INTEGER I;                COMMENT LOCAL COUNTER;
LOGICAL ENTRY;
ENTRY:=TRUE;
FOR I:=1 STEP 1 UNTIL JX-1 DO
   BEGIN
   IF ((FIRST>=BEGIN_OF_PALINDROME(I))
```

```
        AND (LAST<=END_OF_PALINDROME(I))) THEN
        BEGIN
        COMMENT PALINDROME IS ENTIRELY INCLUDED IN PREVIOUSLY RECORDED
              PALINDROME. NO ENTRY REQUIRED;
        ENTRY:=FALSE;
        END
     ELSE
        BEGIN
        IF ((BEGIN_OF_PALINDROME(I) >= FIRST)
           AND (END_OF_PALINDROME(I) <= LAST)) THEN
           BEGIN
           END_OF_PALINDROME(I):=0;
           COMMENT FLAG SMALLER PALINDROME;
           END;
        END;
     END;         COMMENT ALL PREVIOUSLY RECORDED PALINDROMES COMPARED
                          WITH LAST INPUT;
  IF ENTRY = TRUE THEN
     BEGIN
     COMMENT LARGER THAN ALL PREVIOUS OR OVERLAPPING;
     BEGIN_OF_PALINDROME(JX):=FIRST;
     END_OF_PALINDROME(JX):=LAST;
     JX:=JX+1;
     END;
  END RECORD_PALINDROME;

COMMENT MAIN;

INITIALIZE;
READ_AND_WRITE_INPUT_CARDS;
PALINDROME_CHECK;
IF JX = 1 THEN TEXT3
ELSE WRITE_ALL_PALINDROMES;
END.
```

WORKSHEET FOR CODING PHASE OF PROJECT # : 1

Beginning of Coding (day/time) : 1/28/2200

End of Coding (day/time) : 1/31/1220

Man hours : 7.0 (including punching of cards)

CODING BEGIN DAY/TIME	END DAY/TIME	PROGRAM PART	ERROR # [1]	DAY TIME	COMMENT (incl. coded error types)
					1) Record when error is detected.
1/28/2200		-Data Definition -Initialization			
			3	2205	C23
			4	2210	C23
	1/28/2300				
1/30/1335		-Utilities (texts, I/O subroutines)	5	1340	C10
			6	1400	C8
			7	1405	C6
			8	1430	C23
			9	1500	C23
			10	1510	D3
	1/30/1530				
1/30/1615		-Palindrome check	11	1630	C1 Misunderstanding of design
	1/30/1700				
1/30/1700		Coding Review	12	1710	C11
			13	1710	C11 (wrong indentation)
			14	1720	C17
			15	1725	C27
	1/30/1730				
1/31/0930		Punching Cards	16	1030	C15
			17	1050	C4
			18	1110	C20
			19	1115	C28
			20	1125	C12
			21	1125	C19
			22	1130	C24
			23	1135	C12
			24	1140	C24
	1/31/1220				

Remarks:
Man hours spent for punching cards: 2.8

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 1 DEBUG Run # : 1

Begin of Debug Run (day/time) : 1/31/1334

End of Debug Run (day/time) : 2/04/1200

# of Debug Steps incl. in Debug Run: 3 CPU time for Debug run (sec): .02

CPU time for necessary compiles (sec) : 5.31

a) 1.69 b) 1.49 c) 2.13 d) e) f) g)

Man hours for this Debug Run : .9 (excluding overhead)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	All parts	Get errorfree compile	8 compile errors	1/31 1335	.3	25	A1
						26	C5
						27	C12
						28	A2
						29	C23
						30	C12
						31	C11
						32	A1
				1/31 1400			
2	All parts	Get errorfree compile	2 compile errors	1/31 1806	.2	33	C23
						34	A2
				1/31 1830			
3	All parts	Get errorfree compile	O.K.	2/04 1121	.4	35	C28
				2/04 1200			

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 1   DEBUG Run # : 2

Begin of Debug Run (day/time) :   2/05/1330

End of Debug Run (day/time) :   2/05/1500

# of Debug Steps incl. in Debug Run: 6   CPU time for Debug run (sec): .08

CPU time for necessary compiles (sec) : 4.99

a) 2.11   b) 1.46   c) 1.42   d)   e)   f)   g)

Man hours for this Debug Run : 1.7 (excluding overhead)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY/TIME	MAN HOURS/STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	Utilities	Test all Texts exept TEXT3 (all texts should be printed as designed)	Run error	2/05 1345 2/05 1420	.6	36	D11
2	Utilities	Repeat step 1	Run error	2/05 1420 2/05 1445	.5	37 38 39	C8 C27 C20
3	Utilities	Repeat step 1	Spacing problems	2/05 1445	.6	40	C28
4	Utilities	Test spacing and printing of blank lines (all formats should appear as designed)	O.K.				
5	Utilities	Test calculation of number of input cards for minmum length of input string (2 characters) (program should read only one card)	O.K.				
6	Utilities	Test detection and writing of a palindrome of minimum length (palindrome should be detected and listed as designed)	O.K.	2/05 1500			

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 1   DEBUG Run # : 3

Begin of Debug Run (day/time) :   2/05/1500

End of Debug Run   (day/time) :   2/05/1600

# of Debug Steps incl. in Debug Run: 5   CPU time for Debug run (sec): 8.79

CPU time for necessary compiles (sec) : 3.07

a) 1.51   b) 1.56   c)   d)   e)   f)   g)

Man hours for this Debug Run : .9 (excluding overhead)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	1) ERROR #	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	Utilities	Test calculations of number of input cards for maximum string length (i.e. 256 characters) (program should read 4 cards)	O.K.	2/05 1500	.5		
2	Palindrome Check	Test detection and writing of a palidrome of maximum length (PALIDROME COUNTER should be 1, palidrome should be detected and printed as designed)	run error	1500		41	C28
				2/05 1530			
3		Repeat steps 1 and 2	O.K.	2/05 1530	.4		
4	Palindrome check	Test detection and writing of maximum number of palindromes of minimum length (PALINDROME COUNTER should be 128, all 128 palindromes should be listed as designed)	O.K.				
5	Palindrome check	Test for string without palindromes (TEXT3 should be printed)	TEXT3 not printed	1530 1530		42 43	B3 C28
				2/05 1600			

Remarks: "run error" indicates termination due to excess of array limits.

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 1 DEBUG Run # : 4

Begin of Debug Run (day/time) : 2/05/1600

End of Debug Run (day/time) : 2/05/1700

# of Debug Steps incl. in Debug Run: 2 CPU time for Debug run (sec): .22

CPU time for necessary compiles (sec) : 1.67

a) 1.67 b) c) d) e) f) g)

Man hours for this Debug Run : .5 (excluding overhead)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	1) ERROR #	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	Initi-aliza-tion	Test for input of illegal string length (program should print a warning and terminate)	O.K.	2/05 1600	.5		
2	Palin-drome check	Test for overlapping pa-lindromes (should be recorded as two palindromes)	O.K.	2/05 1630			

ERROR LISTING

PROJECT # : 1

Begin of Project (day/time) : 1/28/1040

End of Project (day/time) : 2/06/1620

Man hours for total project : 21.8

ERROR #	PHASE in which ERROR was dis-covered	PHASE in which ERROR was made	ERROR TYPE (see ANNEX F)	TIME spent to solve the ERROR (Man min.)	# of OTHER STATEMENTS OR PARTS OF THE PROGRAM AFFECTED
1	Design	Design	D3	5	Data input only
2	Design	Design	D3	5	Data printout
3	Coding	Coding	C23	5	Indexing within subsequent subroutine
4	Coding	Coding	C23	3	
5	Coding	Coding	C10	2	
6	Coding	Coding	C8	5	
7	Coding	Coding	C6	5	
8	Coding	Coding	C23	5	
9	Coding	Coding	C23	10	3 statements
10	Coding	Design	D3	30	whole subroutine affected
11	Coding	Coding	C1	10	whole subroutine affected
12	Coding	Coding	C11	5	
13	Coding	Coding	C11	3	
14	Coding	Coding	C17	5	
15	Coding	Coding	C27	5	
16	Coding	Coding	C15	5	
17	Coding	Coding	C4	5	
18	Coding	Coding	C20	5	
19	Coding	Coding	C28	5	
20	Coding	Coding	C12	5	
21	Coding	Coding	C19	5	
22	Coding	Coding	C24	10	
23	Coding	Coding	C12	3	
24	Coding	Coding	C24	15	whole subroutine affected
25	Debugging	Coding	A1	10	
26	Debugging	Coding	C5	5	
27	Debugging	Coding	C12	5	

ERROR LISTING

PROJECT # : 1

Begin of Project (day/time) : 1/28/1040

End of Project (day/time) : 2/06/1620

Man hours for total project : 21.8

ERROR #	PHASE in which ERROR was dis-covered	PHASE in which ERROR was made	ERROR TYPE (see ANNEX F)	TIME spent to solve the ERROR (Man min.)	# of OTHER STATEMENTS OR PARTS OF THE PROGRAM AFFECTED
28	Debugging	Coding	A2	5	
29	Debugging	Coding	C23	3	
30	Debugging	Coding	C12	5	
31	Debugging	Coding	C11	3	
32	Debugging	Coding	A1	3	
33	Debugging	Coding	C23	5	
34	Debugging	Coding	A2	5	
35	Debugging	Design	C28	10	
36	Debugging	Design	D11	25	
37	Debugging	Coding	C8	10	Writing of pa-lindromes
38	Debugging	Coding	C27	5	
39	Debugging	Coding	C20	10	Writing of pa-lindromes
40	Debugging	Coding	C28	20	Whole subrou-tine affected
41	Debugging	Coding	C28	15	
42	Debugging	Debugging	B3	3	
43	Debugging	Coding	C28	15	
44	Testing	Design	D9	20	

ERROR LISTING (COMMENTS)

ERROR #	DAY TIME	COMMENTS (EVIDENCE, THOUGHTS, WHY WAS THE ERROR MADE? WHY AND HOW WAS THE ERROR DISCOVERED? ERROR BLOCKING, etc.)
	1/28	
1	1505	Lack of concentration (step left out, which the programmer was aware of)
2	2115	This design step was postponed and forgotten later on.
3	2205	Programmer did not check the manual. (Discovered while examining the solution of a similar problem in an old program.)
4	2210	same as 3
	1/30	
5	1340	
6	1400	
7	1405	
8	1430	Error was found while examining the list of error types in connection with errors 3 and 4.
9	1500	Error was found while reading the programming manual.
10	1510	Errors 10 and 11 discovered during a desk test.
11	1630	see 10
12	1710	(wrong indentation could lead to programming errors)
13	1710	see 12
14	1720	
15	1725	
	1/31	
16	1030	
17	1050	This statement was inserted between the wrong lines. It was detected in this early stage of the project because the programmer was punching cards himself, which allowed him to review his source statements.
18	1110	Programmer did not look at the declaration of the variable.
19	1115	
20	1120	(missing mandatory declaration)
21	1125	(wrong format)
22	1130	
23	1135	same as 20
24	1140	
	1/31	
25	1335	special character (") was expected to be necessary due to similarity with other uses of WRITE/WRITEON, however this assumption was wrong
26	1335	(previously declared variable declared twice)
27	1335	same as 20
28	1335	
29	1335	Programmer did not check the programming manual.
30	1335	see 20
31	1335	
32	1335	

ERROR LISTING (COMMENTS)

ERROR #	DAY TIME	COMMENTS (EVIDENCE, THOUGHTS, WHY WAS THE ERROR MADE? WHY AND HOW WAS THE ERROR DISCOVERED? ERROR BLOCKING, etc.)
	1/31	
33	1806	
34	1806	Error could have been avoided by looking at previous block of code.
	2/04	
35	1122	Porgrammer had correct design concept but did not code correctly.
	2/05	
36	1345	Boundary conditions had not been checked before.
37	1420	
38	1420	
39	1420	
	2/05	
40	1445	Trivial error was made, because programmer was tired.
	2/05	
41	1500	Design was not stated well.
42	1530	same as 40
43	1530	
	2/06	
44	1525	

PROJECT # 1

FINAL STATISTICS

Project name : PALINDROMES

Short description:
Given an input string of length n, where 2 <= n <= 256, find all occurences of palindromes and list them with respect to their beginning and ending positions on the input card.
Input: via punch cards
Output: via line printer

Quantitative measures:

1. # of source statements : 141

2. Total man hours for project : 21.8

3. Man hours spent in

   a) Design : 5.0

   b) Coding : 7.0

   c) Debugging : 4.0

   d) Testing : 5.8

4. CPU time for compiles: 20.22 sec.

5. CPU time for debug runs: 9.11 sec.

6. CPU time for test runs: 13.98 sec.

7. # of test and debug runs: 5

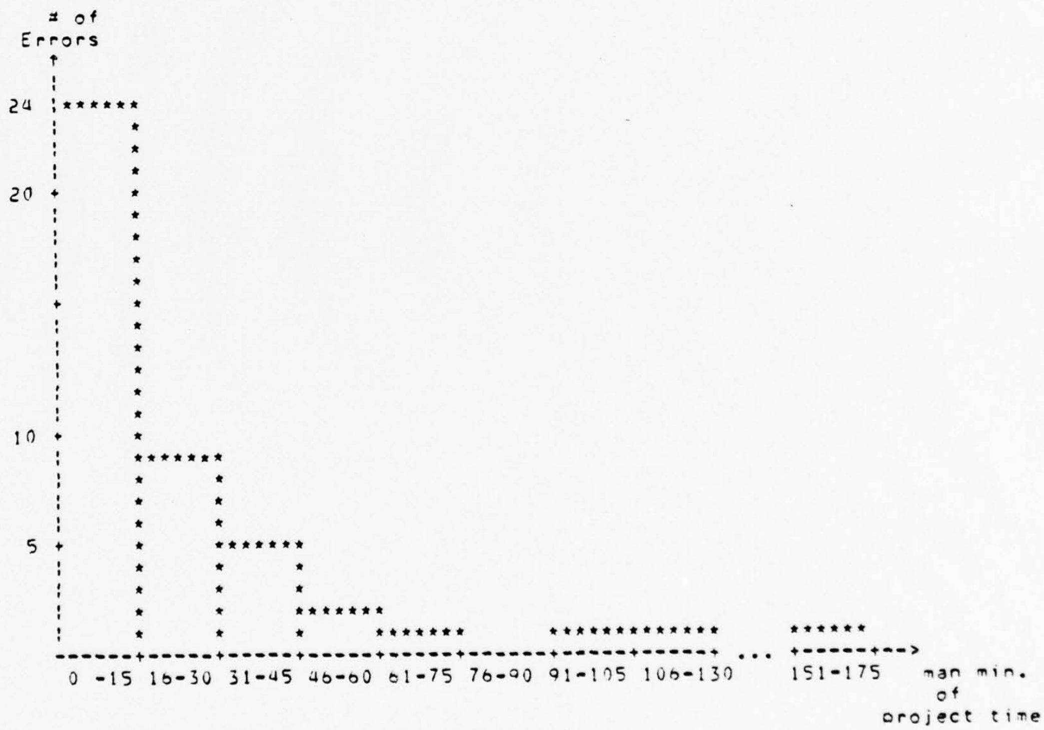
8. # of test and debug steps: 23

9. # of errors found: 44

10. Total man hours used to correct errors: 6.73

11. Error Detection:

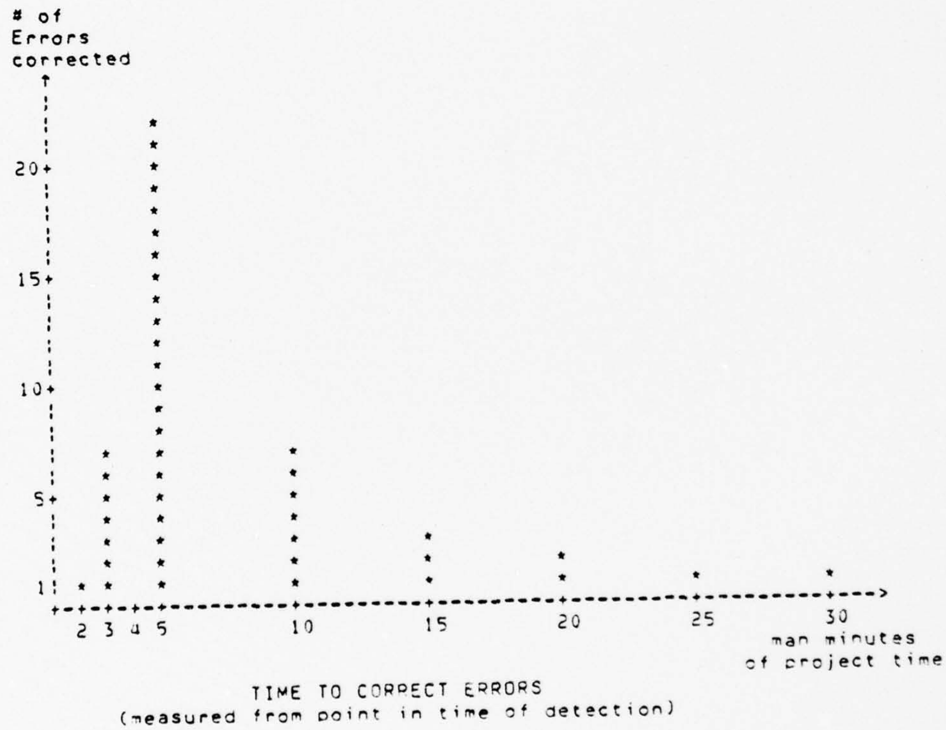
a) Mean time between error detections: 20.3 man min.



TIME BETWEEN ERROR DETECTIONS
(measured from point in time of detection of previous error)

12. Error Correction:

a) Mean time to correct an error: 7.8 man min.



13. When errors were found:

a) # of errors found during design phase:	1 = 2.3 %
b) # of errors found during design review:	1 = 2.3 %
c) # of errors found during coding:	22 = 50.0 %
d) # of errors found during debugging:	19 = 43.2 %
e) # of errors found during writing of test procedures:	0 = 0.0 %
f) # of errors found during testing:	1 = 2.3 %

	44

14. When errors were made:

a) # of errors made during design phase:	5 = 11.4 %
b) # of errors made during design review:	0 = 0.0 %
c) # of errors made during coding:	38 = 86.4 %
d) # of errors made during debugging:	1 = 2.3 %
e) # of errors made during writing of test procedures:	0 = 0.0 %
f) # of errors made during testing:	0 = 0.0 %

	44

15. TIME HISTORY GRAPHS :

```
 # of
 Errors
  ↑
  :
  :                               Error Types:
  :                               A: Clerical error
  :                               B: Debugging error
  :                               C: Coding error
  :                               D: Design error
8 +                        A
  :                        C
  :                        C
5 +                      C A
  :                      C C
  :           C    C     C CC  C
  :         C C    C   CCC CA  C B
1 +    D   DC CCCCCCC  CCC ACDCCCC         D
 -+-+-+-+-+-+---------+---------+---------+----------->
   1 2 3 4 5         10        15        20     man hours
  ..Design...Coding.......Debugging..Testing..
          (Software Development Phases)

       NUMBER OF ERRORS FOUND VS PROJECT TIME
```

```
 # of
 Errors

  ↑
  :
5 +                        *
  :                        *
  :          *           * **
  :         **   **    ***********
  :    *   ***** ****  ************          *
 -+-+-+-+-+-+---------+---------+---------+----------->
   1 2 3 4 5         10        15        20     man hours

     NUMBER OF ERRORS CORRECTED VS PROJECT TIME
```

1. Design Errors

The follwing types of errors apply to both categories "System Design Errors" and "Program Design Errors":

D1 : Communication Error
D2 : Design Negligence
D3 : Forgotten Cases or Steps
D4 : Timing Problems
D5 : Errors in I/O Concepts
D6 : Data Design Error
D7 : Initialization Error
D8 : Inadequate Checking
D9 : Extreme Conditions Neglected
D10: Sequencing Error
D11: Indexing Error
D12: Loop Control Errors
D13: Misuse of Boolean Expression
D14: Mathematical Error
D15: Representation Error
D16: Misunderstanding of Problem Specifications
D17: Other Design Errors

2. Coding Errors

C1 : Misunderstanding of Design
C2 : Negligence
C3 : I/O Format Error
C4 : Misplaced Data Declaration
C5 : Multiple Data Declarations
C6 : Missing Data Declaration
C7 : Inadequate Data
C8 : Initialization Error
C9 : Error in Parameter Passing
C10: Inadequate or Forgotten Checking
C11: Level Problems
C12: Missing Declarations of Block Limits
C13: Case selection error
C14: GO TO Problems
C15: Comment Error
C16: Forgotten Delimiter
C17: Inconsistency in Naming
C18: Wrong Use of Nested IF Statements
C19: Indexing Error
C20: Inconsistent Use of Variables or Data
C21: Sequencing Error
C22: Flag Usage Problems
C23: Syntax Error
C24: Loop Control Error
C25: Incorrect Exit from Subroutines
C26: Language Usage Problems

C27: Forgotten Statements
C28: Representation Error
C29: Control Sequence Error
C30: Incorrect Subroutine Usage
C31: Other Coding Errors

3. Clerical Errors

A1 : Manual Error
A2 : Mental Error
A3 : Procedural Errors
A4 : Other Clerical Errors

4. Debugging Errors

B1 : Inappropriate Use of Debugging Tools
B2 : Insufficient or Inappropriate Selection of Test Cases or Test Data
B3 : Misinterpretation of Debugging Results
B4 : Misinterpretation of Error Source
B5 : Negligence
B6 : Other Debugging Errors

5. Testing Errors

T1 : Inadequate Test Case(s) or Test Data
T2 : Misinterpretation of Test Results
T3 : Misinterpretation of Program Specification
T4 : Negligence
T5 : Other Testing Errors

DIRECTED GRAPH REPRESENTATION

PROJECT # : 1

Program part : INITIALIZATION

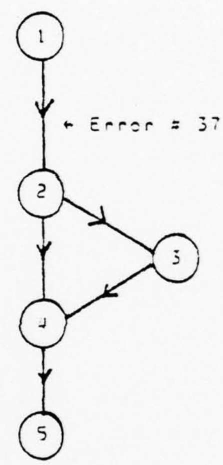
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 14

NUMBER OF NODES: 5
NUMBER OF ARCS : 5

NUMBER OF PATHS: 2
CYCLOMATIC NUMBER: V(G)= 2

REACHABILITY OF NODES:
NODE 1 : 1
NODE 2 : 1
NODE 3 : 1
NODE 4 : 2
NODE 5 : 2
SUM: 7.000000
REACHABILITY INDEX
OF DIRECTED GRAPH: 1.400000



DIRECTED GRAPH REPRESENTATION

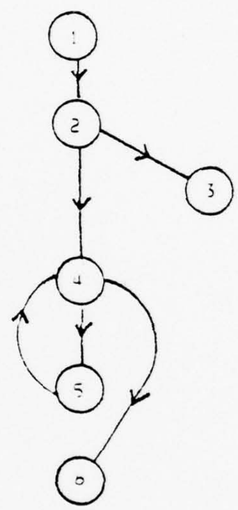
PROJECT # : 1

Program part : BLANK LINES

COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 6

```
NUMBER OF NODES:     6
NUMBER OF ARCS :     6
NUMBER OF PATHS:     3
CYCLOMATIC NUMBER:  V(G)=     2

REACHABILITY OF NODES:
NODE     1  :     1
NODE     2  :     1
NODE     3  :     1
NODE     4  :     2
NODE     5  :     1
NODE     6  :     2
              ---------
     SUM:       8.000000
REACHABILITY INDEX
OF DIRECTED GRAPH:         1.333333
```



DIRECTED GRAPH REPRESENTATION

PROJECT # : 1

Program part : TEXT1, TEXT2, TEXT3

COMPLEXITY MEASURES:

```
NUMBER OF STATEMENTS: 4 - 11
NUMBER OF NODES:      2
NUMBER OF ARCS :      1
NUMBER OF PATHS:      1
CYCLOMATIC NUMBER:  V(G)=    1

REACHABILITY OF NODES:
NODE    1  :     1
NODE    2  :     1
              ---------
    SUM:      2.000000
REACHABILITY INDEX
OF DIRECTED GRAPH:       1.000000
```

ANNEX G

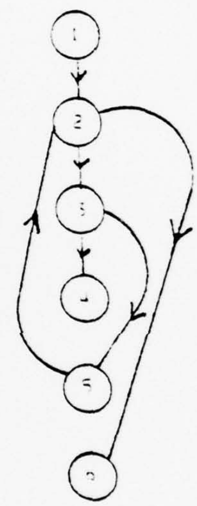
DIRECTED GRAPH REPRESENTATION

PROJECT # : 1

Program part : READ AND WRITE INPUT CARDS

COMPLEXITY MEASURES:

```
NUMBER OF STATEMENTS: 18
NUMBER OF NODES:    6
NUMBER OF ARCS :    7
NUMBER OF PATHS:    3
CYCLOMATIC NUMBER:  V(G)=    3

REACHABILITY OF NODES:
NODE     1  :     1
NODE     2  :     6
NODE     3  :     6
NODE     4  :     2
NODE     5  :     5
NODE     6  :     6
                ---------
     SUM:        26.00000
 REACHABILITY INDEX
 OF DIRECTED GRAPH:          4.333333
```



DIRECTED GRAPH REPRESENTATION

PROJECT # : 1

Program part : WRITE ALL PALINDROMES

a) # of nodes: 15

b) # of arcs : 19

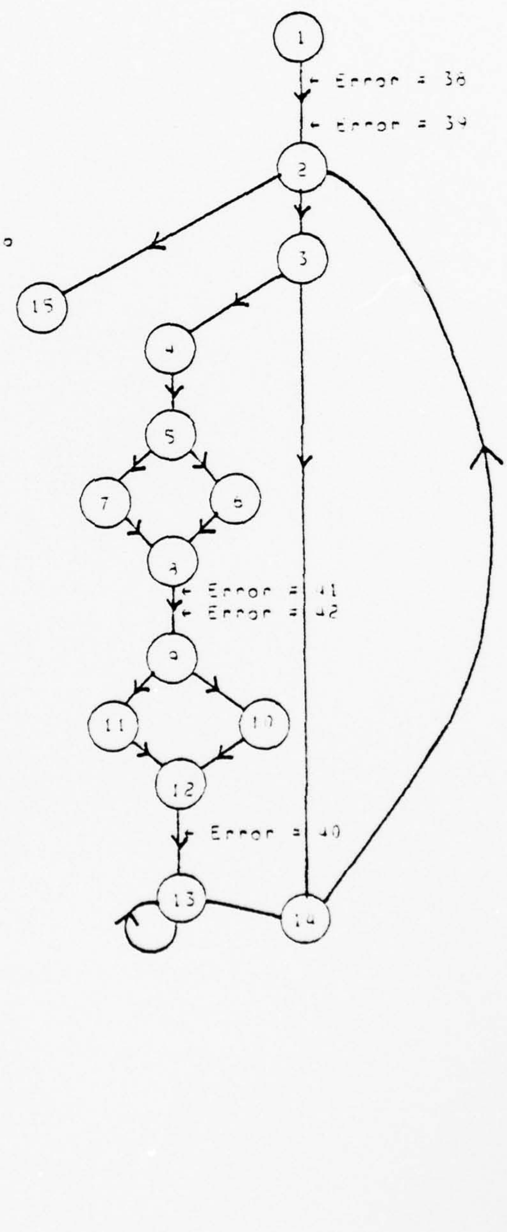
c) # of statements: 26

d) # of paths: *

e) Reachability: *

f) Cyclomatic number: 6

* Number of paths and reachability are very large.



DIRECTED GRAPH REPRESENTATION

PROJECT # : 1

Program part : PALINDROME CHECK

a) # of nodes: 10

b) # of arcs : 13

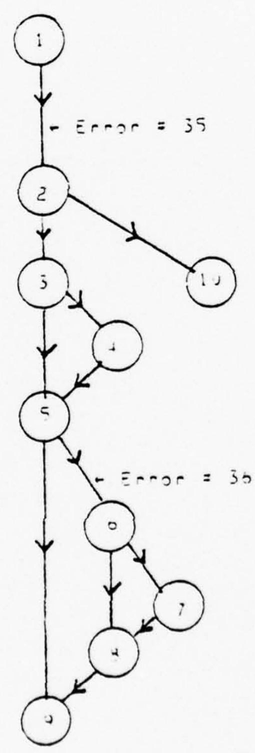
c) # of statements: 7

d) # of paths: *

e) Reachability: *

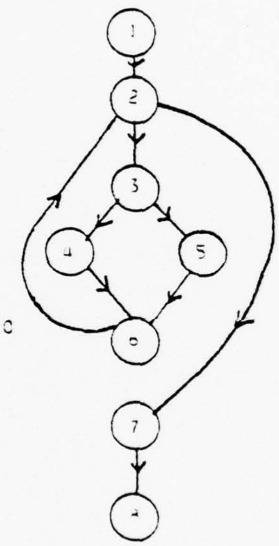
f) Cyclomatic number: 5

* Number of paths
and reachability
are very large.



DIRECTED GRAPH REPRESENTATION

```
PROJECT # :  1

Program part : CONTINUE CHECKING


COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 15
NUMBER OF NODES:     8
NUMBER OF ARCS :     9
NUMBER OF PATHS:     7
CYCLOMATIC NUMBER:  V(G)=     3

REACHABILITY OF NODES:

NODE     1  :     1
NODE     2  :     7
NODE     3  :     6
NODE     4  :     3
NODE     5  :     3
NODE     6  :     6
NODE     7  :     7
NODE     8  :     7
                ---------
      SUM:       40.00000
REACHABILITY INDEX
OF DIRECTED GRAPH:        5.000000
```



DIRECTED GRAPH REPRESENTATION

PROJECT # : 1

Program part : RECORD PALINDROME

a) # of nodes: 9

b) # of arcs : 12

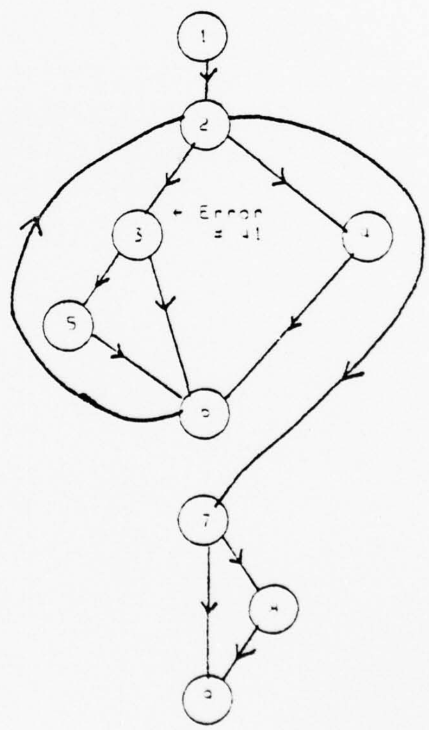
c) # of statements: 21

d) # of paths: *

e) Reachability: *

f) Cyclomatic number: 5

* Number of paths
and reachability
are very large.

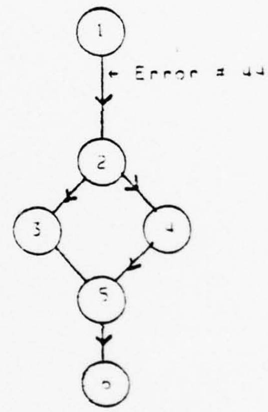


DIRECTED GRAPH REPRESENTATION

PROJECT # : 1

Program part : MAIN

```
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 6

NUMBER OF NODES:      6
NUMBER OF ARCS :      6
NUMBER OF PATHS:      2
CYCLOMATIC NUMBER:  V(G)=     2

REACHABILITY OF NODES:
NODE     1  :     1
NODE     2  :     1
NODE     3  :     1
NODE     4  :     1
NODE     5  :     2
NODE     6  :     2
               ---------
     SUM:       8.000000
REACHABILITY INDEX
OF DIRECTED GRAPH:          1.333333
```



TEST PHASE DESCRIPTION

Project # : 1

Test run # : 1   Including 7 Test Steps

Begin of Test (day time) : 2/06/1430   End of Test (day/time) : 2/06/1620

CPU time for necessary compiles (in sec.): 5.18
a) 1.69 b) 1.71 c) 1.78 d)   e)   f)   g)

CPU time for TEST run (sec) : 13.98

Man Hours in Testing : 5.8 (including preparation of tests)

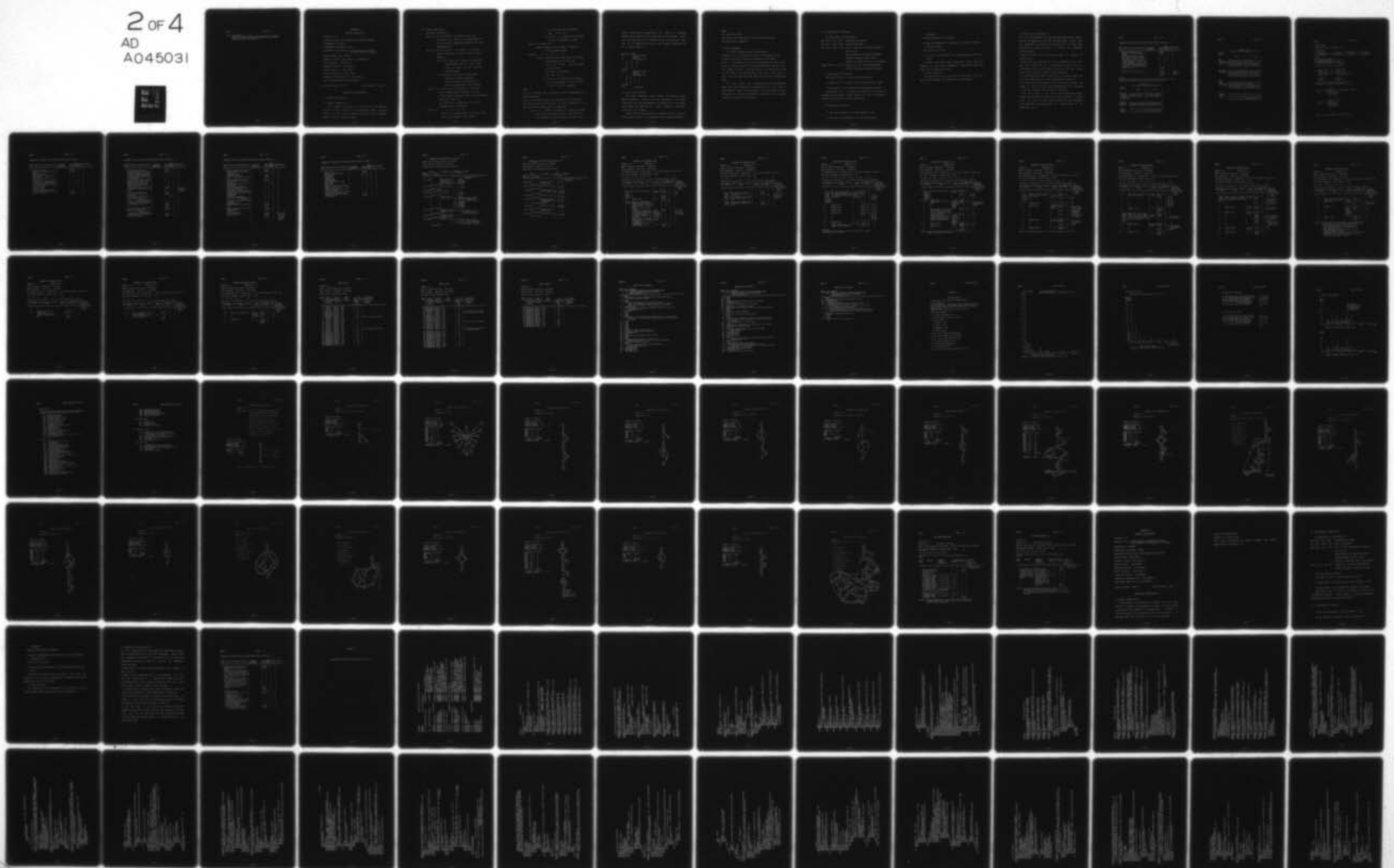
TEST STEP	OBJECTVE	EXPECTED RESULT (TOLERANCE)	ACTUAL RESULT	1) ERROR #	DAY TIME	COMMENTS AND CODED ERROR TYPES
						1) Record when error occurs.
1	Check program for small palindromes (a) palindrome of length 2 (b) palindrome of length 3	All palindromes are detected and recorded as specified by program specifications	O.K.		2/06 1430	
2	Check for large palidromes	same as 1				
	(a) palindrome of length 255		progr. error	44	1525	D9
	(b) palindrome of length 256					
3	Test correction Repeat steps 1,2	same as 1	job terminated		1550	Implementation error (time estimate exceeded)
4	Test correction Repeat steps 1,2	same as 1	O.K.			
5	Check palindromes which cross card boundaries	should be dected and printed as specified	O.K.			
6	Check for palindromes of various lengths and sizes	same as 5	O.K.			
7	Check for invalid input (string length = 1)	Warning and termination	O.K.		2/06 1620	See notes

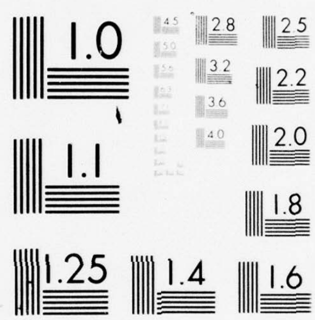
Notes: Other invalid inputs were exhaustively tested during debugging phase.

AD-A045 031 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
AN EXPERIMENT IN SOFTWARE ERROR OCCURRENCE AND DETECTION.(U)
JUN 77 H HOFFMANN

UNCLASSIFIED NL

2 OF 4
AD
A045031





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Choice of Test Data:

Some test data were chosen to check the program for boundary conditions. Other test data were selected to check random size and numbers of palindromes.

APPENDIX B

PROJECT DESCRIPTION

Project # : 2

Project title: PATH ANALYSIS IN DIRECTED GRAPHS

Programmer : HOFFMANN

Programming Language : ALGOL

Programming environment: IBM/360/67,OS/MVT,BATCH
and TIME SHARING (CP/CMS)

Design notes : see ANNEX A

Program listing : see ANNEX B of APPENDIX C

Coding notes : see ANNEX C

Debugging notes : see ANNEX D

Error Listing : see ANNEX E

Final statistics : see ANNEX F

Graphical representation : see ANNEX G

Test phase description: see ANNEX H

Starting date: 1 MAR 77 Ending date: 17 MAR 77

EXPERIMENT DESCRIPTION

1. Project description:

A. General Description

This program is designed to find and write all possible paths of minimal length (i.e. if a directed graph contains loops, no loop may be traversed more than once in succession) through a directed graph.

B. Input Description:

(a) Batch Processing :

Card 1: 1 in card column 1 followed by any text

(this text will identify the graph and will
be printed as a headline preceding the input
documentation)

Card 2: t in card column 1 followed by one blank,

where t specifies the type of path listing
desired.

t=0 -> all paths are listed during analysis
(recommended only for a very large
number of paths, i.e. more than 500
paths expected)

t is nonzero -> paths are listed after
analysis has been completed
(any additional information will
not be interpreted by the program)

Card 3: s in card column 1 followed by one blank

where s specifies the number of digits
of the largest node name used in the directed
graph and 1<= s <=3
(any additional information will not be
interpreted by the program)

Card 4: 1 n s1 s2 ... sn

where 1 is the node name of the entry node
and may be punched in any column

n is the number of successors

and 0 <= n <= 14

si (i=1,2, ... n) specifies the name

of the ith sucessor of node 1

Card 5 thru Card m+3:

where m specifies the number of nodes in

in the directed graph

Format: j n s1 s2 ... sn

where j specifies the node name of the node

being described on this card

and 2<= j <= 999; j may be punched

in any column.

n is number of successors

and 0<= n <= 14

si (i=1,2 ... n) specifies the name of

the iths successor (1 <= si <= 999)

if n=0 -> no entry necessary

NOTE:

(a) All entries are integers and must be separated by at least one blank.

(b) All information about any node must be on one card.

(c) Card column 80 must contain a blank character or an additional blank card has to be inserted.

Card 4 + m: 99999 followed by at least one blank

(this input indicates that all information of

the preceeding directed graph has been input)

NOTE: Following this description any number of directed graphs may be specified and submitted to the program. After the last description of a directed graph a termination card has to be added which contains any integer different from "1" in card column 1.

Sample Input:

```
1            (FIRST GRAPH)
0            TOGGLE
2            MAX NODE NAME
     1     1     2
     2     2     3     4
     3     0
     4     1     5
     5     0
99999
1            (SECOND GRAPH)
1            TOGGLE
1            MAX NODE NAME
     1    2    3    2
     2    0
     3    1    4
     4    2    1    5
     5    0
99999
-1            TERMINATE
```

(b) Input under CP/CMS:

The input sequence under CP/CMS is similar as under batch processing. There is only one exception: Any other input than the input requested by the program will terminate the program. The user can only input integers, additional blanks are not allowed.

Under CP/CMS instructions are displayed at the terminal which makes the use of the program almost self-explanatory.

NOTE:

(a) Login with 540k

(b) Follow the directions of the program precisely.

(c) Input only integers.

C. Error Messages:

All error messages are self-explanatory.

Errors 3, 8, 9, 10 refer to invalid or incomplete input.

Errors 8 and 9 will cause the program to terminate.

Error 2 indicates the limit of the program. If error 2 occurs it is most likely that the directed graph has an infinite number of paths. The user may try the same input using "0" as input on the second card. Error 1 will only occur if more than 30 nodes have been specified having 14 successors each.

All other errors will indicate an error which has not been found during testing and debugging or will be due to some abnormal usage of the program. In either case it might be possible to locate the problem by examining the program listing or retrying the program.

2. Programmer's background:

a) Experience in programming:

Oct 1970 - May 1971	Programming courses
May 1971 - April 1972	Module Programmer
May 1972 - June 1974	Work in Test and Simulation Department at the NAVAL COMMAND AND CONTROL SYSTEMS COMMAND (FEDERAL GERMAN NAVY) Testing of tactical real time systems
March 1975 - Jan 1977	Student at the NAVAL POSTGRADUATE SCHOOL, Monterey, Computer Science

b) Experience in testing:

Two years of work in testing and simulation.

c) Experience in the area of the given problem: None.

d) Experience in the progamming language being used:

Experience over a period of 18 months in more than 10 programming projects. (Total number of source statements produced during that time was more 4000.)

3. Psychological factors:

a) Did the programmer like the project? Yes.

b) How does the programmer like the programming

language?

Favorite programming language.

c) Was the programmer satisfied by the way the problem was specified?

Only minor criticism.

d) How did the programmer like the programming environment?

The facilities (study room, card punch room) were not conducive to efficient programming because of restricted space, bad lighting and noise.

e) Other factors:

The recording of the experiment's data during the project affected speed and concentration considerably.

4. Comments on Documentation

For the documentation of each software development phase a special documentation form has been developed. These forms are designed to provide a firm guideline for the experiment programmer to record all data of interest for subsequent error analysis.

- Begin and end of each step was recorded with respect to day and time.
- Each error was recorded when it is discovered. The error was then identified by a unique error number (1,2,...). Furthermore the time of discovery and the error type (using error types listed in ANNEX F) were recorded.
- If appropriate, comments about error discovery, reason why the error was made, etc. were documented in ANNEX E2.
- For each error the phase in which the error was made, the phase in which the error was discovered and the time spent to correct the error was recorded in ANNEX E1.
- For each step in any one of the software development phases the day/time of begin and end was recorded. In addition, the time (in man hours) for each step was recorded. This excludes the overhead used for documentation of the experiment data.

WORKSHEET FOR DESIGN PHASE AND DESIGN REVIEW PHASE OF PROJECT # 2

STEP #	PROBLEM AND PLANNED SOLUTION	ALTERNATE SOLUTIONS	DAY TIME	MAN HOURS /STEP	ERROR #	COMMENT
1	Analyzing structures with respect to minimum number of paths. The minimum number of paths includes all possible paths starting at the entry node (1) and ending at any node which has no successor. The minimum number excludes all paths having repetative traversals of loops, such that any sequence of arcs is traversed more than once in a row.		3/01 1900 3/03 2400	12.0		
2	Design of Data Structures:		3/04 1100 3/04 1900			Details see Remarks

Remarks:
Two freelists are used to supply storage space to keep information about paths and the structure of the underlying directed graph.

FREELIST 1 (M1)
(451 Items of 4 bytes each (32 Bits))

Items used for:	< 2 Bytes >	< 2 Bytes >
a) Successor Information (element in forward linked list)	successor name	ptr next successor
b) Header of successor list	reachability parent	ptr successor list
c) Freelist Header (Item 0)	0	ptr next free item

FREELIST 2 (M2)
[20001 Items of 8 Bytes each (64 Bits)]

Items used for:	< 2 Bytes >	< 2 Bytes >
a) Node Information (duplicate)	pointer to original	node name
	ptr to successor	ptr to predecessor
b) Pathheader Information	ptr to previous path	path identification
	ptr to begin of path	ptr to next path
c) Node Node Information (original)	pointer to original	node name
	# of successors	ptr successor list
d) Freelist Header (Item 0)	0	0
	0	ptr next free item

Typical usage:

a) Original node with successor list

```
    |-----|-----|
    |    p|    i|
    |-----|-----|      |-----|-----|      |-----|-----|          |-----|-----|
p   |    n|ptr---------->|    r|ptr------>|    j|ptr--------->|    k| NIL |
    |-----|-----|      |-----|-----|      |-----|-----|          |-----|-----|
```

Where:
i, j, k are node names
r represents reachability of node i
p is the storage location within M2
NIL indicates the end of a linked list

b) Doubly linked list of path headers:

```
      |-----|-----|                 |-----|-----|
<-----ptr  |  i  |  <---------------ptr  | i+1 |
      |-----|-----|                 |-----|-----|
      | ptr | ptr----------->       | ptr | ptr----------->
      |--|--|-----|                 |--|--|-----|
         |                             |
         V                             V
     beginning                     beginning
     of path i                    of path i+1
```

Where i and i+1 are path identifications.
c) Nodes (duplicates) being elements of a path

```
                              .
                              .
                       |-----|-----|
original  <-----------ptr|    i |   ↑ predecessor of node i
  of i                 |-----|-----|  |
                       | ptr |ptr-----'
                       |--|--|-----|
                          |     ↑
                          |     |......
                          |           |
                          V           |
                       |-----|-----|  |
original  <-----------ptr|    j |   ↑
  of j                 |-----|-----|  |
                       |  ptr|ptr-----'
                       |--|--|-----|
                          |
                          |
                          V
                              .
                              .
```

Where i and j are successive nodes on a path.

WORKSHEET FOR DESIGN PHASE AND DESIGN REVIEW PHASE OF PROJECT # 2

STEP #	PROBLEM AND PLANNED SOLUTION	ALTERNATE SOLUTIONS	DAY TIME	MAN HOURS /STEP	ERROR #	COMMENT
3	TOP - DOWN Design:		3/05			
	Define the following program		1500			
	parts:					
	-Data structures					
	-Primitives to support data					
	structures					
	-Utilities to support data					
	structures					
	-Logical procedures to support			1.0		
	analysis of directed graphs					
	-Subroutines to support analy-					
	sis of possible paths through					
	a directed graph					
	-Input of data					
	-Output of data					
	-Error messages and relevant					
	diagnostics to support error					
	error investigation		3/05			
			1600			

WORKSHEET FOR DESIGN PHASE AND DESIGN REVIEW PHASE OF PROJECT # 2

STEP #	PROBLEM AND PLANNED SOLUTION	ALTERNATE SOLUTIONS	DAY TIME	MAN HOURS /STEP	ERROR #	COMMENT
4	Define Data Structures		3/05			
	a) Use Freelist (M1) of 451		1600			
	4 byte words used to provide			.5		
	information about successors					
	of parent nodes (each of max.					
	30 parent nodes may have up to					
	14 successors)					
	b) Use Freelist (M2) of 20001					
	8 byte items to keep informa-					
	about original nodes, path					
	headers and possible paths					
	through the directed graph					
	being analyzed.					
	(each item is designed to hold		3/05			
	4 integer values between 0 and		1630			see also
	65535)					remarks
						on page 1
5	Design of primitives and		3/05			
	utilities to support data		1630			
	structures					
	a) Primitives for Freelist 1:			5.5		
	-Initialize Freelist 1					
	-Car1, Cdr1 (retrieval of integer values)					
	-Setcar1, Setcdr1 (set values)					
	-Allocate1, Free1 (allocate/					
	free elements of M1)		3/05			
			2030			
	b) Primitives for Freelist 2:		3/06			
	-Initialize Freelist 2		1330			
	-Car2, Cdr2 (retrieval of integer values)					
	-Setcar2, Setcdr2 (set values)					
	-Allocate2, Free2 (allocate/		3/06			
	free elements of M2)		1500			

WORKSHEET FOR DESIGN PHASE AND DESIGN REVIEW PHASE OF PROJECT # 2

STEP #	PROBLEM AND PLANNED SOLUTION	ALTERNATE SOLUTIONS	DAY TIME	MAN HOURS /STEP	ERROR #	COMMENT
6	Define Algorithms for Utilities to support structures using M1: -Name, Setname -Brother, Addbrother -Son(i) (get name of ith son)		3/06 1800 3/06 1830	.5		
7	Define Algorithms for Utilities to support structures using M2: a)Support of linked list of path headers: -Set Path ID, -Path ID -Linkpath(forward/backward) -Initialize Pathlist -Next path, -Previous Path b) Support of path structures: -Name of (retrieve node name) -Predecessor -Successor -Number of Successors -Linking of nodes(forward and backward) -Duplicate(set duplicate node) -Addnode -Duplicate a Path -Implementation of alternative path		3/07 1000 3/07 1200	2.0		
8	Define Procedural Algorithms: -Remove a Path -List Path -List all Paths -Find End of a Path -Find Original		3/08 1500 1505 3/08 1530	.5	17	D9 former analysis did not include trivial case

WORKSHEET FOR DESIGN PHASE AND DESIGN REVIEW PHASE OF PROJECT # 2

STEP #	PROBLEM AND PLANNED SOLUTION	ALTERNATE SOLUTIONS	DAY TIME	MAN HOURS /STEP	ERROR #	COMMENT
9	Design Review:		3/09			
	Define global pointers:		1100			
	-Pathhead					
	-Currentpath			3.0		
	-Lastpath					
	-Currentnode					
	-Lastoccurence					
	Define algorithm for path analysis:					
	-Set Paths					
	Specify logical algorithms:					
	-Occurs Twice (node with same name on same path)					
	-One Way (loop without alternative branches)					
	-Match (same sequence of nodes duplicated on current path)					
	-Checkback (check for invalid alternatives)		3/09 1400			

WORKSHEET FOR CODING PHASE OF PROJECT # : 2

Beginning of Coding (day/time) : 3/06/1500

End of Coding (day/time) : 3/10/1730

Man hours : 26 (including punching of cards)

CODING BEGIN DAY/TIME	CODING END DAY/TIME	PROGRAM PART	ERROR # [1]	DAY TIME	COMMENT
					1) Record when error is detected.
03/06/1500		-Data Definition		3/06	
		-Primitives	1	1530	C28
		-Error Handling	2	1540	C28
		-Initialization	3	1620	C23
	03/06/1630				
03/06/1630		-Punching cards			
			4	1740	C28
	03/06/1800				
03/06/1830		-Utilities			
	03/06/2030				
03/07/1200		-Utilities		3/07	
			5	1305	D9
	03/07/1400				
03/08/1000		-Punching cards		3/08	
		(Utilities)	10	1000	C12 (missing BEGIN)
			11	1005	C28 (mispelling of procedure name)
			12	1005	C23
			13	1010	C16
			14	1025	C1
			15	1030	C17
			16	1030	C17
	03/08/1200				
03/08/1530		-Coding of procedural subroutines	18	1630	D9 (design did not consider removal of first and last path)
	03/08/1800				
03/08/1900		-Coding of procedural subroutines			
	03/08/2000				
03/09/2000		-Punching cards			
			19	2130	D11 (faulty design of index calculation)
			20	2150	C10 (faulty condition "Not Equal" instead of "="
	03/08/2200				

WORKSHEET FOR CODING PHASE OF PROJECT # : 2

Beginning of Coding (day/time) : 3/06/1500

End of Coding (day/time) : 3/10/1730

Man hours : 26 (including punching of cards)

CODING BEGIN DAY/TIME	END DAY/TIME	PROGRAM PART	ERROR # [1]	DAY TIME	COMMENT
					1) Record when error is detected.
03/09/1200		-Coding of I/O subroutines		3/09	
	03/09/1300				
03/09/1300		-Punching cards			
			34	1440	C11
	03/09/1500				
03/09/1500		-Coding of Logical procedures	35	1530	D9
			36	1615	D9
	03/09/1730				
03/09/2000		-Punching cards			
			40	2030	C20
	03/09/2100				
03/09/2100		-Coding of path analyzing algorithm("Set paths")	42	2200	D9
	03/09/2300				
03/10/1000		-Punching cards		3/10	
			44	1040	D15
	03/10/1100				
03/10/1100		-Coding of I/O subroutines			
	03/10/1200				
03/10/1630		-Punching cards			
			46	1700	D9
	03/10/1730				

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 2 DEBUG Run # : 1

Begin of Debug Run (day/time) : 03/06/1800

End of Debug Run (day/time) : 03/07/1400

# of Debug Steps incl. in Debug Run: 4 CPU time for Debug run (sec): 1.82

CPU time for necessary compiles (sec) : 4.33

a) 0.89 b) 1.76 c) 1.68 d) e) f) g)

Man hours for this Debug Run : 4.0 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	Primitives and part of utilities	Get error free compile	3 compile errors	3/06 1811	1.3	5 6 7	A1 C23 A1
				3/06 1930			
2		Repeat step 1	O.K.	3/07 1110	1.2		
3	Primitives	Check initialization, allocation and freeing of items in both freelists. Check parameter implementation. Check writing of blank lines. (pointers of freelists must be set appropriately, allocated elements must be filled with zeroes, allocated items must be unlinked, after freeing an element becomes part of the freelist again, parameters should be set as designed)	upper half of allocated element of M2 not filled with zeroes	1250	0.5	8	A1 ("+1" left out while punching cards
4		repeat steps 1 and 3	O.K.	3/07 1400	1.0		

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 2   DEBUG Run # : 2

Begin of Debug Run (day/time) : 03/08/1000

End of Debug Run (day/time) : 03/08/1700

# of Debug Steps incl. in Debug Run: 2   CPU time for Debug run (sec): .46

CPU time for necessary compiles (sec) : 2.23

a) 2.23   b)   c)   d)   e)   f)   g)

Man hours for this Debug Run : 2.5 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	Primi- tives and part of utili- ties	Check maximum number of allocations in both free-lists. (all elements except free-list headers are alloca- ted)	O.K.	3/08 1000	0.3		2 man hours spent in preparation of debug run
2	Error hand- ling	Check error messages for exhaustion of each free-list.	O.K.	3/08 1700	0.2		

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 2 DEBUG Run # : 3

Begin of Debug Run (day/time) : 03/08/2200

End of Debug Run (day/time) : 03/09/1300

# of Debug Steps incl. in Debug Run: 7 CPU time for Debug run (sec): .6

CPU time for necessary compiles (sec) : 18.12

a) 2.67 b) 2.75 c) 2.75 d) 2.43 e) 3.49 f) 4.03 g)

Man hours for this Debug Run : 4.5 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	Primitives, Error handling, Procedural subroutines	Get error free compile of all subroutines and primitives excluding I/O subroutines and logical subroutines.	75 compile diagnostics (see remarks)	3/08 2200	1.0		
				2200		21	C23
						22	C17
						23	C27
						24	A1
2		Repeat step 1		2245	0.5	25	C17
3		Repeat step 1		2310	0.5	26	A2
4		Repeat step 1		2323	0.5	27	C28
5		Repeat step 1		3/09 1030	1.0	28	A1
						29	A1
						30	A1
						31	A1
						32	A1
						33	C27
6		Repeat step 1	O.K.	1130	0.3		
7		Repeat debug runs 1 and 2 (same results expected)	O.K.	1150 3/09	1.2		

Remarks:
The ALGOL compiler diagnostics do not always allow a complete investigation of all errors.

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 2   DEBUG Run # : 4

Begin of Debug Run (day/time) : 03/09/1500

End of Debug Run   (day/time) : 03/10/1200

# of Debug Steps incl. in Debug Run: 6   CPU time for Debug run (sec): .61

CPU time for necessary compiles (sec) : 23.45

a) 3.67   b) 5.02   c) 5.20   d) 4.60   e) 4.96   f)   g)

Man hours for this Debug Run : 3.0 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	All subroutines except logical procedures	Get error free compile.	2 compile errors	3/09 1500	0.5		
				1730		37	C23
						38	A1
2		Repeat step 1	O.K.	1730			
3	I/O	Check input for trivial cases	number of successors placed in wrong field	1950	0.5		
				2000		39	C26 (READ and READON confused)
4		Repeat step 3 with trace and preceding initialization of freelists as well as initialization of list of path headers.		2130	0.5		
		Check function "SON" for correct return values (trace should match with design considerations)	wrong value returned from "Number of Successors"	2200		41	A1
5		Repeat step 4	Missing link	2320	0.5		
				2340		43	C9
6		Repeat step 4	O.K.	2350	0.5		
				3/10 1100			

Remarks: Functions and subroutines are first checked for trivial cases and boundary conditions.

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 2   DEBUG Run # : 5

Begin of Debug Run (day/time) : 03/10/1600

End of Debug Run (day/time) : 03/10/2400

# of Debug Steps incl. in Debug Run: 5   CPU time for Debug run (sec): 1.35

CPU time for necessary compiles (sec) : 29.49

a) 5.71   b) 5.94   c) 5.43   d) 5.79   e) 6.62   f)   g)

Man hours for this Debug Run : 5.0 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # [1]	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	All Primitives and utilities	Check for trivial inputs for all primitives and utilities.	1 error in linking	3/10 1530	0.5		
				1600		45	C27
2		Repeat step 1	1 error	1730	1.0	47	D9
3		Repeat step 1	3 program errors found	1930	2.0		
				1930		48	C21 (statements in reverse order)
				2100		49	D12 (wrong stopping condition)
				2130		50	C8 (faulty initialization of loop)
4		Repeat step 1	2 program errors	2200	1.0		
				2250		51	B4 (error in correcting).
				2300		52	C24
5		Repeat step 1	O.K.	3/10 2330	0.5		

Remarks: Procedures checked for trivial cases and boundary conditions.

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 2 DEBUG Run # : 6

Begin of Debug Run (day/time) : 03/10/2230

End of Debug Run (day/time) : 03/11/1700

# of Debug Steps incl. in Debug Run: 5 CPU time for Debug run (sec): 2.14

CPU time for necessary compiles (sec) : 26.3

a) 5.14 b) 6.62 c) 7.38 d) 7.16 e) f) g)

Man hours for this Debug Run : 6.0 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR 1) #	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	All	Get error free compile of all program parts.	8 compile errors	3/10 2300	1.0		
				2310		53	A1
						54	A1
						55	A1
						56	A1
						57	A1
						58	C28
						59	C9
						60	C9
2		Repeat step 1	O.K.	2330 3/10 2400	0.5		
3	Logical procedures and writing of a path	Check for trivial cases while analyzing one path. Check writing of a path. (all functions should perform as designed)	1 error found	3/11 1400 1400	3.0	61	C27 (reset forgotten)
4		Repeat step 3	2 errors found	1545 1620	1.0	62	C10 (wrong stopping condition)
				1630		63	C30
5		Repeat step 3	O.K.	1630 3/11 1700	0.5		

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 2 DEBUG Run # : 7

Begin of Debug Run (day/time) : 03/11/1700

End of Debug Run (day/time) : 03/12/1600

# of Debug Steps incl. in Debug Run: 5 CPU time for Debug run (sec): 2.61

CPU time for necessary compiles (sec) : 38.78

a) 8.05 b) 7.81 c) 7.88 d) 7.82 e) 7.22 f) g)

Man hours for this Debug Run : 7.0 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	Procedure "Set paths"	Check analysis of pathes. (All pathes are implemented indoubly linked lists)	2 program errors	3/11 1700	0.5		
				1730		64	D12 (wrong stopping condition)
				1745		65	D11 (wrong indexing)
2		Repeat step 1	1 error found	1805	0.5		
				1810		66	D11 (same as error 65)
3		Repeat step 1	5 errors found	1820	3.5		
				3/12 1300		67	D3 (forgotten
				1315		68	D3 update
				1320		69	D3 of a variable)
				1325		70	C29
				1340		71	B4 (misinterpretation of error source)
4		Repeat step 1	1 error found	1400	1.5	72	D9
5		Repeat step 1	O.K.	1530 3/12 1600	0.5		

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 2 DEBUG Run # : 8

Begin of Debug Run (day/time) : 03/13/1600

End of Debug Run (day/time) : 03/14/1600

# of Debug Steps incl. in Debug Run: 3 CPU time for Debug run (sec): 162.75

CPU time for necessary compiles (sec) : 19.59

a) 6.63 b) 6.45 c) 6.51 d) e) f) g)

Man hours for this Debug Run : 17.0 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	All	Check various small directed graphs for boundary conditions.	1 error found in "Matching"	3/13 1600 2000	8.0	73	C29 (wrong branching)
2		Repeat step 1	1 error found (some paths skipped during analysis)	3/14 0700 1100	6.0	74	D10
3		Repeat step 1	O.K.	1300 3/14 1600	3.0		

Remarks: Error analysis during this debug run was extremely difficult and time consuming for the following reasons:

- Most results were verified by desk checking.
- The implementation of paths was checked by examining the contents of allocated items of both freelists (M1,M2).
- Some structures being tested had a very large number of paths (> 2000).
- The program contains a great number of rather complex algorithms, each of which had to be tested with respect to its boundary conditions.

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 2   DEBUG Run # : 9

Begin of Debug Run (day/time) : 03/13/1100

End of Debug Run (day/time) : 03/15/2300

# of Debug Steps incl. in Debug Run: 1   CPU time for Debug run (sec): 3.92

CPU time for necessary compiles (sec) : 7.52

a) 7.52   b)   c)   d)   e)   f)   g)

Man hours for this Debug Run : 3.0 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	All	Check internal error messages. (Error messages 5,6,7, 11,12,13 should be prin-ted)	O.K.	3/14 2345 - 3/15 2300	3.0		

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 2 DEBUG Run # : 10

Begin of Debug Run (day/time) : 03/16/1500

End of Debug Run (day/time) : 03/16/1600

# of Debug Steps incl. in Debug Run: 1 CPU time for Debug run (sec): 6.75

CPU time for necessary compiles (sec) : 7.73

a) 7.73 b) c) d) e) f) g)

Man hours for this Debug Run : 1.0 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR[1] #	COMMENTS AND CODED ERROR TYPES
1	All	Check some user dependent error messages. (error messages 10 and 8 should be printed)	O.K.	3/16 1500 3/16 1600	1.0		1) Record when error occurs

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 2 DEBUG Run # : 11

Begin of Debug Run (day/time) : 03/16/1600

End of Debug Run (day/time) : 03/17/1200

# of Debug Steps incl. in Debug Run: 2 CPU time for Debug run (sec): 6.45

CPU time for necessary compiles (sec) : 14.95

a) 7.90 b) 7.05 c) d) e) f) g)

Man hours for this Debug Run : 2.0 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	A11	Check error message # 9.	error message # 9 not printed	3/16 1600 1800	1.0	75	C21
2		Repeat step 1	O.K.	3/17 1140 3/17 1200	1.0		

ERROR LISTING

PROJECT # : 2

Begin of Project (day/time) : 03/01/1900

End of Project (day/time) : 03/17/2200

Man hours for total project : 125.0

ERROR #	PHASE in which ERROR was dis- covered	PHASE in which ERROR was made	ERROR TYPE (see ANNEX F)	TIME spent to solve the ERROR (Man min.)	# of OTHER STATEMENTS OR PARTS OF THE PROGRAM AFFECTED
1	Coding	Coding	C28	5	
2	Coding	Coding	C28	5	
3	Coding	Coding	C23	2	
4	Coding	Coding	C28	1	
5	Debugging	Coding	A1	5	
6	Debugging	Coding	C23	5	
7	Debugging	Coding	A1	5	
8	Debugging	Coding	A1	10	
9	Coding	Design	D9	15	Whole algorithm affected.
10	Coding	Coding	C12	1	
11	Coding	Coding	C28	1	
12	Coding	Coding	C23	1	
13	Coding	Coding	C16	1	
14	Coding	Coding	C1	5	
15	Coding	Coding	C17	2	
16	Coding	Coding	C17	2	
17	Design	Design	D9	1	
18	Coding	Design	D9	30	whole algorithm affected.
19	Coding	Design	D11	5	
20	Coding	Coding	C10	5	
21	Debugging	Coding	C23	5	
22	Debugging	Coding	C17	15	
23	Debugging	Coding	C27	5	
24	Debugging	Coding	A1	2	
25	Debugging	Coding	C17	5	
26	Debugging	Coding	A2	10	
27	Debugging	Coding	C28	10	
28	Debugging	Coding	C17	5	
29	Debugging	Coding	C11	5	
30	Debugging	Coding	A1	15	

ERROR LISTING

PROJECT # : 2

Begin of Project (day/time) : 03/01/1900

End of Project (day/time) : 03/17/2200

Man hours for total project : 125.0

ERROR #	PHASE in which ERROR was dis- covered	PHASE in which ERROR was made	ERROR TYPE (see ANNEX F)	TIME spent to solve the ERROR (Man min.)	# of OTHER STATEMENTS OR PARTS OF THE PROGRAM AFFECTED
31	Debugging	Coding	A1	5	
32	Debugging	Coding	A1	2	
33	Debugging	Coding	C27	5	
34	Coding	Coding	C11	1	Whole algorithm affected.
35	Coding	Design	D9	15	Whole algorithm affected.
36	Coding	Design	D9	15	
37	Debugging	Coding	C23	5	
38	Debugging	Coding	A1	5	
39	Debugging	Coding	C26	30	
40	Debugging	Coding	C20	5	
41	Debugging	Coding	A1	5	
42	Coding	Design	D9	20	Whole algorithm affected.
43	Debugging	Design	D9	10	
44	Coding	Design	D15	5	
45	Debugging	Coding	C27	10	
46	Coding	Design	D9	5	
47	Debugging	Design	D9	60	Whole subroutine affected. (3 changes necessary)
48	Debugging	Coding	C21	10	
49	Debugging	Design	D12	5	
50	Debugging	Coding	C8	15	
51	Debugging	Debugging	B4	20	
52	Debugging	Coding	C24	5	
53	Debugging	Coding	A1	5	
54	Debugging	Coding	A1	2	
55	Debugging	Coding	A1	2	
56	Debugging	Coding	A1	2	
57	Debugging	Coding	A1	2	
58	Debugging	Coding	C28	5	
59	Debugging	Coding	C9	5	
60	Debugging	Coding	C9	5	

ERROR LISTING

PROJECT # : 2

Begin of Project (day/time) : 03/01/1900

End of Project (day/time) : 03/17/2200

Man hours for total project : 125.0

ERROR #	PHASE in which ERROR was dis-covered	PHASE in which ERROR was made	ERROR TYPE (see ANNEX F)	TIME spent to solve the ERROR (Man min.)	# of OTHER STATEMENTS OR PARTS OF THE PROGRAM AFFECTED
61	Debugging	Coding	C27	30	
62	Debugging	Coding	C10	10	
63	Debugging	Coding	C29	10	
64	Debugging	Design	D12	10	
65	Debugging	Design	D11	5	
66	Debugging	Design	D11	15	
67	Debugging	Design	D3	10	
68	Debugging	Design	D3	10	
69	Debugging	Design	D3	5	
70	Debugging	Coding	C30	15	
71	Debugging	Debugging	B4	10	
72	Debugging	Design	D9	120	
73	Debugging	Coding	C29	180	
74	Debugging	Design	D10	60	
75	Debugging	Coding	C21	30	

ERROR LISTING (COMMENTS)

ERROR #	DAY TIME	COMMENTS (EVIDENCE, THOUGHTS, WHY WAS THE ERROR MADE? WHY AND HOW WAS THE ERROR DISCOVERED? ERROR BLOCKING, etc.)
	03/06	
1	1530	Errors 1,2,3 were discovered while reading previously written sections of code.
2	1540	
3	1620	
4	1740	Lack of concentration while punching cards. (Main disadvantage while punching cards is that punched data is not immediately seen after each key stroke.)
5	1815	same as 4
6	1815	Programmer did not check programming manual. (error could have been avoided)
7	1815	same as 4
	03/07	
8	1305	
9	1305	
	03/08	
10	1000	Errors 10-17 were detected because programmer punched cards himself. It is quite natural that he uses this time to review his code.
11	1005	
12	1005	
13	1010	
14	1025	
15	1030	
16	1030	
17	1505	
18	1630	Error found during desk test.
19	2130	Error found while punching cards.
20	2150	same as 19
21	2200	
22	2200	Function name used as local variable.
23	2200	
24	2200	
25	2245	Incomplete correction of error # 22.
26	2310	Mandatory declaration omitted while punching cards.
27	2323	Right parentesis omitted.
	03/09	
28	1030	Lack of concentration while punching cards. (Programmer was tired.)
29	1030	same as 28
30	1030	same as 28
31	1030	same as 28
32	1030	same as 28

ERROR LISTING (COMMENTS)

ERROR #	DAY TIME	COMMENTS (EVIDENCE, THOUGHTS, WHY WAS THE ERROR MADE? WHY AND HOW WAS THE ERROR DISCOVERED? ERROR BLOCKING, etc.)
	03/09	
33	1030	
34	1440	
35	1530	Extreme conditions not carefully analyzed.
36	1615	same as 35
37	1730	Lack of concentration while coding.
38	1730	
39	2000	"READ" and "READON" confused.
40	2030	Using a local variable name from different subroutine.
41	2200	
42	2230	
43	2340	Use of wrong parameter.
	03/09	
44	1040	Error found while punching cards.
	03/10	
45	1600	
46	1700	Thinking of boundary conditions while punching cards.
47	1730	Desk test was not made carefully enough. (one case omitted)
48	1930	Two statements in reverse order.
49	2100	
50	2130	Error was discovered during Structured Walk Through.
51	2250	Caused by changing code.
52	2300	
53	2310	Lack of concentration while punching cards. (see comment # 4)
54	2310	same as 53
55	2310	same as 53
56	2310	same as 53
57	2310	same as 53
58	2310	
59	2310	Programmer did not check previous procedure declaration. (faulty invocation)
60	2310	same as 59
	03/11	
61	1400	Forgotten reset of parameter.
62	1620	Inappropriate stopping condition in loop. (confused with similar condition in a previuos problem)
63	1630	Wrong use of a subroutine.
64	1730	Error found using a trace.
65	1745	same as 64
66	1810	Error found during desk test.

ERROR LISTING (COMMENTS)

ERROR #	DAY TIME	COMMENTS (EVIDENCE, THOUGHTS, WHY WAS THE ERROR MADE? WHY AND HOW WAS THE ERROR DISCOVERED? ERROR BLOCKING, etc.)
	03/12	
67	1300	Necessary update of a variable left out.
68	1315	same as 67
69	1320	same as 67
70	1325	Boolean expression does not meet the needs of the algorithm. Error found by tracing.
71	1340	Misinterprtation of the error source. Error found while repeating debugging step.
72	1400	Different behaviour of algoritm for removal of last path previous to the last one was not considered. Error found by desk test of all relevant subroutines.
	03/14	
73	2000	Error found during desk test.
	03/15	
74	1100	
	03/16	
75	1800	Error found during desk test.

PROJECT # 2

FINAL STATISTICS

Project name : PATH ANALYSIS IN DIRECTED GRAPHS

Short description:
This program is designed to find all possible paths of minimal length (i.e. if a directed graph contains loops no loop may be traversed more than once in a row) through a directed graph.
Input: via punch cards Output: via line printer

Quantitative measures:

1. # of source statements : 712

2. Total man hours for project : 125.0

3. Man hours spent in

   a) Design : 31.0

   b) Coding : 26.0

   c) Debugging : 55.0

   d) Testing : 13.0

4. CPU time for compiles: 206.85 sec.

5. CPU time for debug runs: 189.46 sec.

6. CPU time for test runs: 869.73 sec.

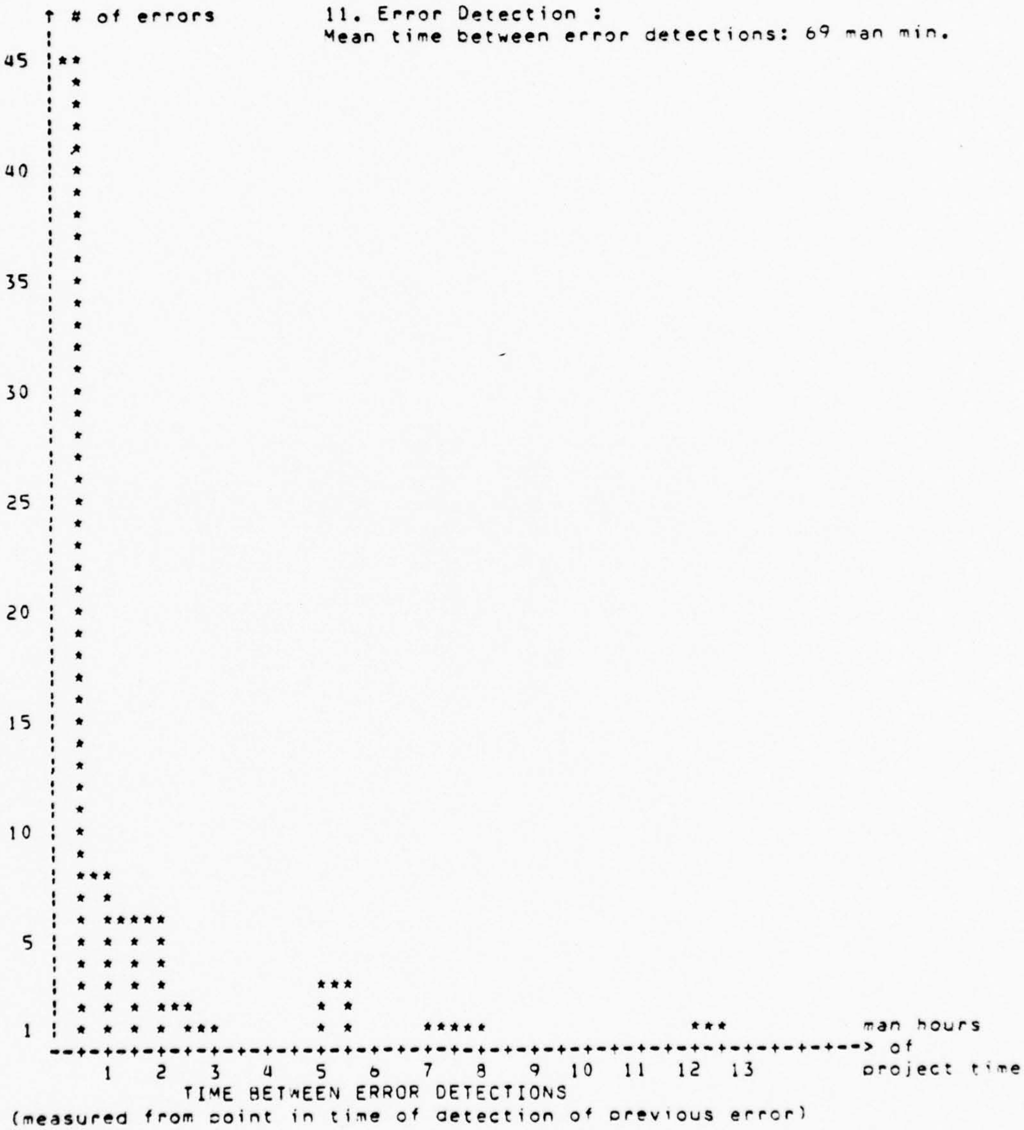
7. # of test and debug runs: 13

8. # of test and debug steps: 43

9. # of errors found: 75

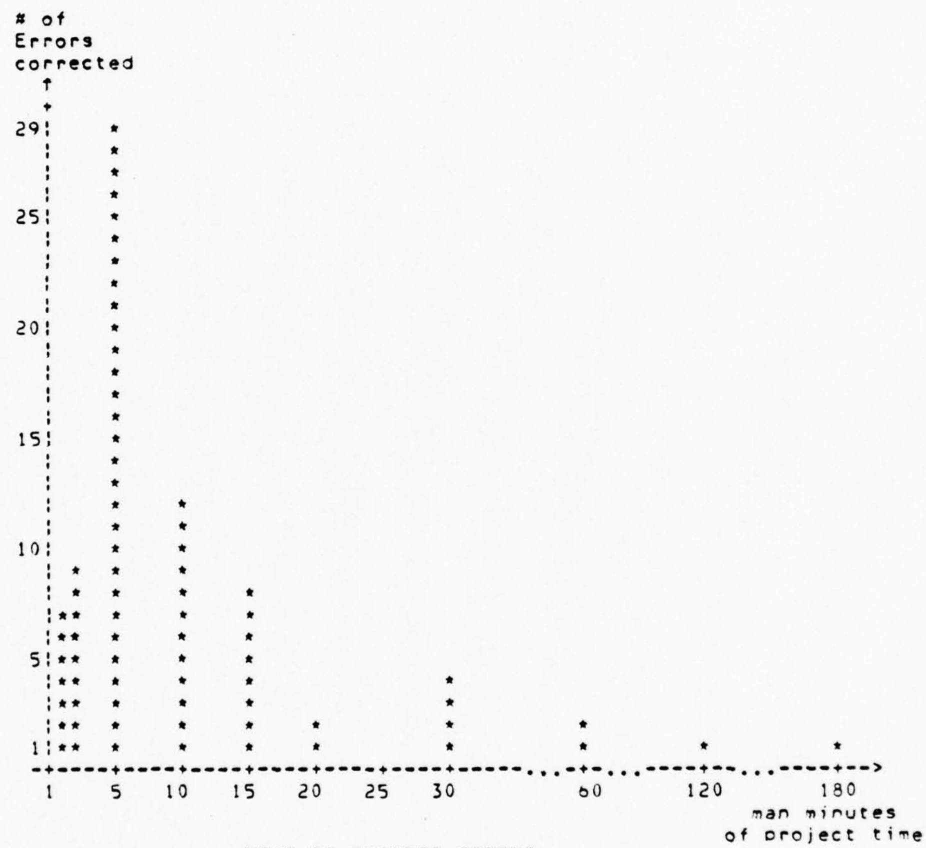
10. Total man hours used to correct errors: 16.5

11. Error Detection :
Mean time between error detections: 69 man min.



12. Error Correction:

Mean time to correct an error: 13.2 man min.



TIME TO CORRECT ERRORS
(measured in from point in time of detection)

13. When errors were found:

a) # of errors found during design phase:	1 = 1.3 %
b) # of errors found during design review:	0 = 0.0 %
c) # of errors found during coding:	21 = 28.0 %
d) # of errors found during debugging:	53 = 70.7 %
e) # of errors found during writing of test procedures:	0 = 0.0 %
f) # of errors found during testing:	0 = 0.0 %

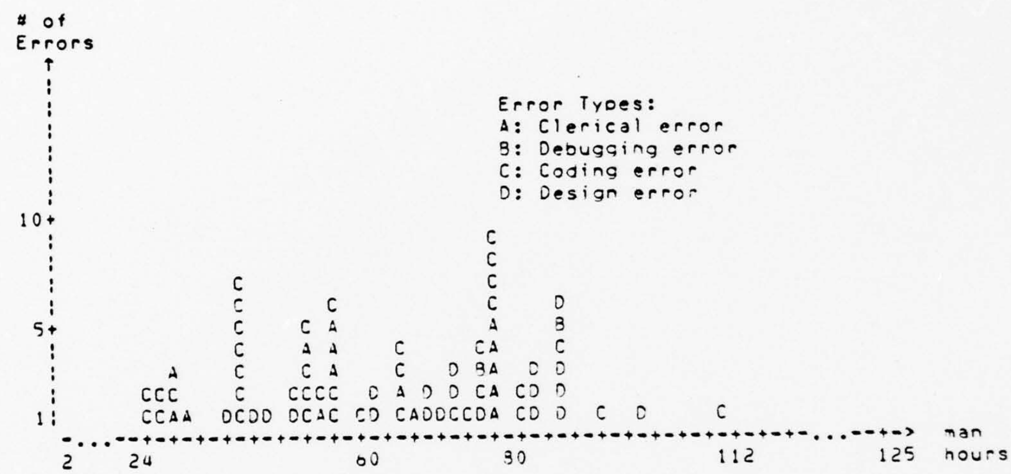
	75

14. When errors were made:

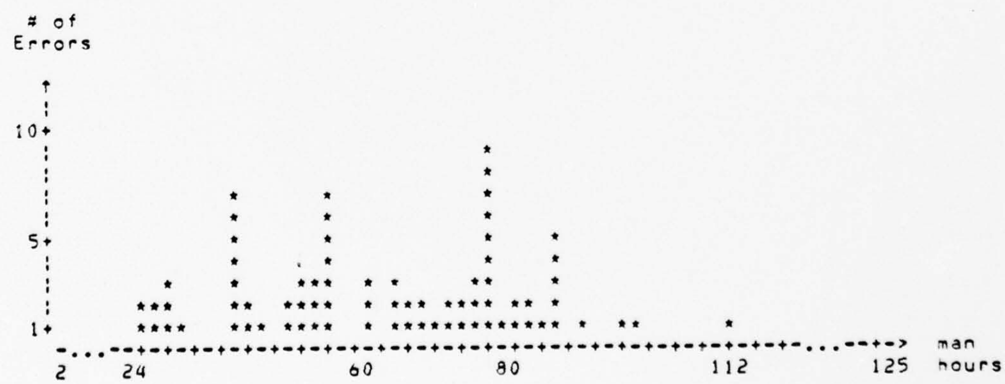
a) # of errors made during design phase:	20 = 26.7 %
b) # of errors made during design review:	0 = 0.0 %
c) # of errors made during coding:	53 = 70.7 %
d) # of errors made during debugging:	2 = 2.7 %
e) # of errors made during writing of test procedures:	0 = 0.0 %
f) # of errors made during testing:	0 = 0.0 %

	75

15. TIME HISTORY GRAPHS :



NUMBER OF ERRORS FOUND VS PROJECT TIME



NUMBER OF ERRORS CORRECTED VS PROJECT TIME

1. Design Errors

The follwing types of errors apply to both categories "System Design Errors" and "Program Design Errors":

D1 : Communication Error
D2 : Design Negligence
D3 : Forgotten Cases or Steps
D4 : Timing Problems
D5 : Errors in I/O Concepts
D6 : Data Design Error
D7 : Initialization Error
D8 : Inadequate Checking
D9 : Extreme Conditions Neglected
D10: Sequencing Error
D11: Indexing Error
D12: Loop Control Errors
D13: Misuse of Boolean Expression
D14: Mathematical Error
D15: Representation Error
D16: Misunderstanding of Problem Specifications
D17: Other Design Errors

2. Coding Errors

C1 : Misunderstanding of Design
C2 : Negligence
C3 : I/O Format Error
C4 : Misplaced Data Declaration
C5 : Multiple Data Declarations
C6 : Missing Data Declaration
C7 : Inadequate Data
C8 : Initialization Error
C9 : Error in Parameter Passing
C10: Inadequate or Forgotten Checking
C11: Level Problems
C12: Missing Declarations of Block Limits
C13: Case selection error
C14: GO TO Problems
C15: Comment Error
C16: Forgotten Delimiter
C17: Inconsistency in Naming
C18: Wrong Use of Nested IF Statements
C19: Indexing Error
C20: Inconsistent Use of Variables or Data
C21: Sequencing Error
C22: Flag Usage Problems
C23: Syntax Error
C24: Loop Control Error
C25: Incorrect Exit from Subroutines
C26: Language Usage Problems

C27: Forgotten Statements
C28: Representation Error
C29: Control Sequence Error
C30: Incorrect Subroutine Usage
C31: Other Coding Errors

3. Clerical Errors

A1 : Manual Error
A2 : Mental Error
A3 : Procedural Errors
A4 : Other Clerical Errors

4. Debugging Errors

B1 : Inappropriate Use of Debugging Tools
B2 : Insufficient or Inappropriate Selection of Test Cases or Test Data
B3 : Misinterpretation of Debugging Results
B4 : Misinterpretation of Error Source
B5 : Negligence
B6 : Other Debugging Errors

5. Testing Errors

T1 : Inadequate Test Case(s) or Test Data
T2 : Misinterpretation of Test Results
T3 : Misinterpretation of Program Specification
T4 : Negligence
T5 : Other Testing Errors

DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : CAR1, CAR2, CDR1,CDR2, SETCAR1,SETCAR2, SETCDR1, SETCDR2, FREE1, FREE2, SPACE, INITIALIZE ALL, SET NAME, ADDBROTHER, BROTHER, NAME OF, PATH ID, NAME, NUMBER OF SUCCESSORS, PREDECESSOR, SUCCESSOR, SET PATH ID, PUT NEXT PATH-HEADER, NEXT PATH, PREVIOUS PATH, LINK PATH BACK, LINKPATH, LINK FORWARD, LINKBACK, DUPLICATE, ADDNODE, ALTERNATIVE WAY, INITPATHLIST

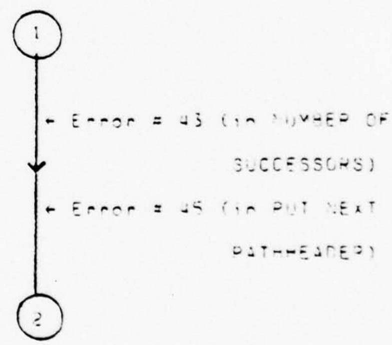
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 3 - 17

NUMBER OF NODES: 2
NUMBER OF ARCS : 1
NUMBER OF PATHS: 1
CYCLOMATIC NUMBER: V(G)= 1

REACHABILITY OF NODES:
NODE 1 : 1
NODE 2 : 1

SUM: 2.000000



NOTE: All 33 subroutines have the same structure.

DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : ALLOCATE1, ALLOCATE2

COMPLEXITY MEASURES:

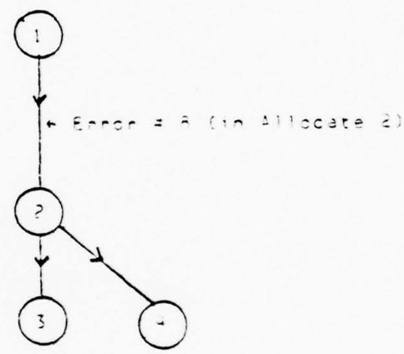
NUMBER OF STATEMENTS: 7 - 8

NUMBER OF NODES: 4
NUMBER OF ARCS : 3
NUMBER OF PATHS: 2
CYCLOMATIC NUMBER: V(G)= 1

REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	1
NODE	3 :	1
NODE	4 :	1
SUM:		4.000000

REACHABILITY INDEX
OF DIRECTED GRAPH: 1.000000

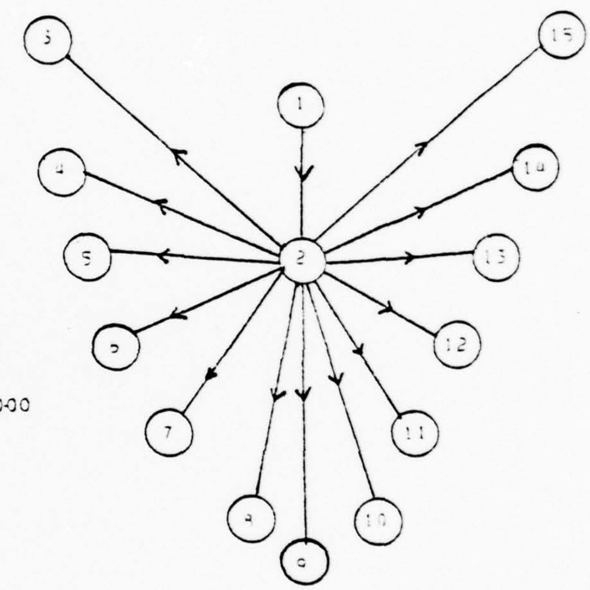


DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : ERROR

```
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 64
NUMBER OF NODES:    15
NUMBER OF ARCS :    14
NUMBER OF PATHS:    13
CYCLOMATIC NUMBER:  V(G)=     1

REACHABILITY OF NODES:
NODE      1  :    1
NODE      2  :    1
NODE      3  :    1
NODE      4  :    1
NODE      5  :    1
NODE      6  :    1
NODE      7  :    1
NODE      8  :    1
NODE      9  :    1
NODE     10  :    1
NODE     11  :    1
NODE     12  :    1
NODE     13  :    1
NODE     14  :    1
NODE     15  :    1
                ---------
     SUM:          15.00000
REACHABILITY INDEX
OF DIRECTED GRAPH:          1.000000
```



DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : DIAGNOSE

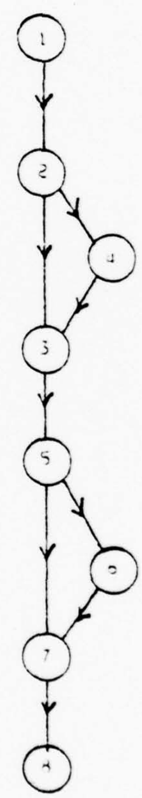
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 32
NUMBER OF NODES: 8
NUMBER OF ARCS : 9
NUMBER OF PATHS: 4
CYCLOMATIC NUMBER: V(G)= **3**

REACHABILITY OF NODES:

NODE 1	:	1
NODE 2	:	1
NODE 3	:	2
NODE 4	:	1
NODE 5	:	2
NODE 6	:	2
NODE 7	:	4
NODE 8	:	4
SUM:		17.00000

REACHABILITY INDEX
OF DIRECTED GRAPH: 2.125000



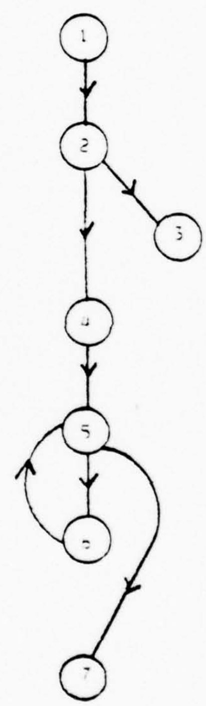
DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : SON

COMPLEXITY MEASURES:

```
NUMBER OF STATEMENTS: 15
NUMBER OF NODES:      7
NUMBER OF ARCS :      7
NUMBER OF PATHS:      3
CYCLOMATIC NUMBER:  V(G)=     2

REACHABILITY OF NODES:
NODE    1  :   1
NODE    2  :   1

NODE    3  :   1
NODE    4  :   1
NODE    5  :   2
NODE    6  :   1
NODE    7  :   2
              ---------
     SUM:      9.000000
REACHABILITY INDEX
OF DIRECTED GRAPH:       1.285714
```



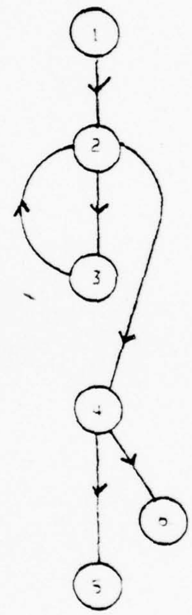
DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : FIND ORIGINAL

COMPLEXITY MEASURES:

```
NUMBER OF STATEMENTS: 11
NUMBER OF NODES:     6
NUMBER OF ARCS :     6
NUMBER OF PATHS:     4
CYCLOMATIC NUMBER:  V(G)=     2

REACHABILITY OF NODES:
NODE     1  :   1
NODE     2  :   2
NODE     3  :   1
NODE     4  :   2
NODE     5  :   2
NODE     6  :   2
               ---------
     SUM:       10.00000
REACHABILITY INDEX
OF DIRECTED GRAPH:        1.666666
```



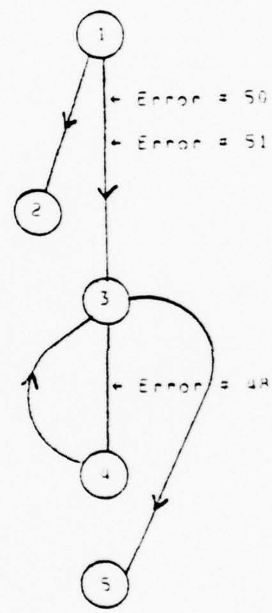
DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : DUPLICATE CURRENT PATH

COMPLEXITY MEASURES:

```
NUMBER OF STATEMENTS: 15
NUMBER OF NODES:     5
NUMBER OF ARCS :     5
NUMBER OF PATHS:     3
CYCLOMATIC NUMBER:  V(G)=     2

REACHABILITY OF NODES:
NODE    1  :    1
NODE    2  :    1
NODE    3  :    2
NODE    4  :    1
NODE    5  :    2
              ---------
     SUM:      7.000000
REACHABILITY INDEX
OF DIRECTED GRAPH:        1.400000
```



DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : FIND PATH

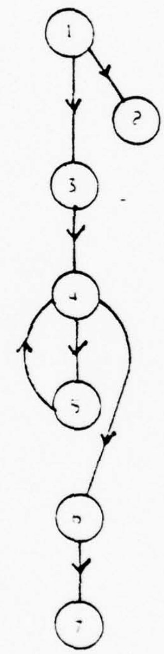
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 12
NUMBER OF NODES: 7
NUMBER OF ARCS : 7
NUMBER OF PATHS: 3
CYCLOMATIC NUMBER: V(G)= 2

REACHABILITY OF NODES:

NODE 1	:	1
NODE 2	:	1
NODE 3	:	1
NODE 4	:	2
NODE 5	:	1
NODE 6	:	2
NODE 7	:	2
SUM:		10.00000

REACHABILITY INDEX
OF DIRECTED GRAPH: 1.428571



DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : REMOVE PATH

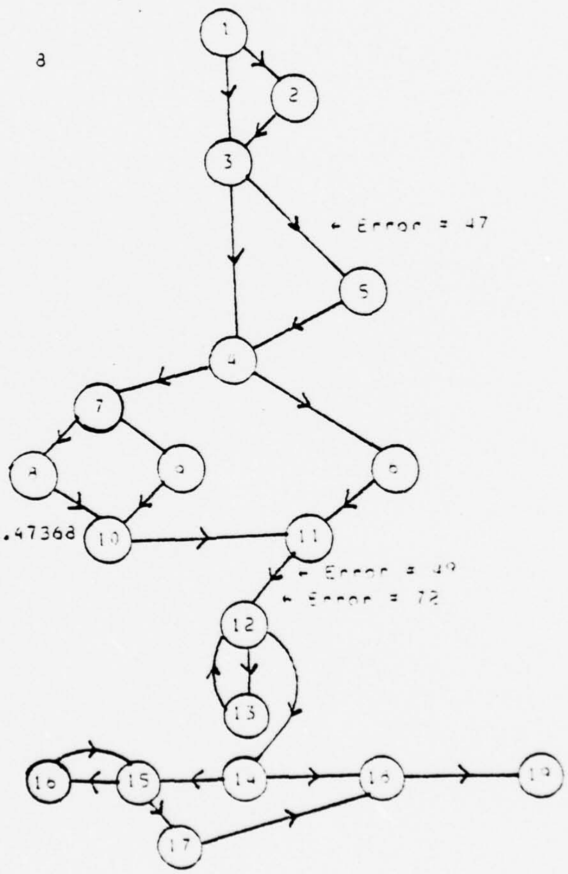
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 45
NUMBER OF NODES: 19
NUMBER OF ARCS : 25
NUMBER OF PATHS: 72
CYCLOMATIC NUMBER: V(G)= 8

REACHABILITY OF NODES:

NODE	1	: 1
NODE	2	: 1
NODE	3	: 2
NODE	4	: 4
NODE	5	: 2
NODE	6	: 4
NODE	7	: 4
NODE	8	: 4
NODE	9	: 4
NODE	10	: 8
NODE	11	: 12
NODE	12	: 24
NODE	13	: 12
NODE	14	: 24
NODE	15	: 48
NODE	16	: 24
NODE	17	: 48
NODE	18	: 72
NODE	19	: 72
SUM =		370.0000

REACHABILITY INDEX
OF DIRECTED GRAPH: 19.47368



DIRECTED GRAPH REPRESENTATION

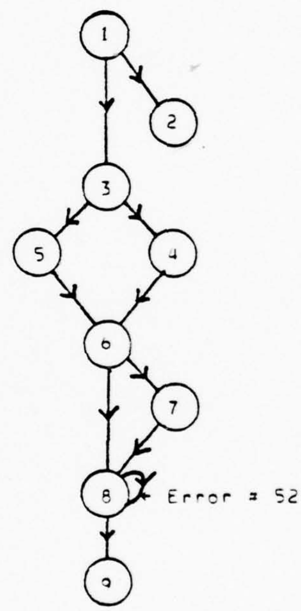
PROJECT # : 2

Program part : END OF PATH

COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 18

```
NUMBER OF NODES:      9
NUMBER OF ARCS :     11
NUMBER OF PATHS:      9
CYCLOMATIC NUMBER:  V(G)=     4

REACHABILITY OF NODES:
NODE      1  :    1
NODE      3  :    1
NODE      2  :    1
NODE      4  :    1
NODE      5  :    1
NODE      6  :    2
NODE      7  :    2
NODE      8  :    8
NODE      9  :    8
                ---------
    SUM:         25.00000
REACHABILITY INDEX
OF DIRECTED GRAPH:         2.777777
```



DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : INPUT DIRECTED GRAPH INFORMATION

a) # of nodes: 23

b) # of arcs : 27

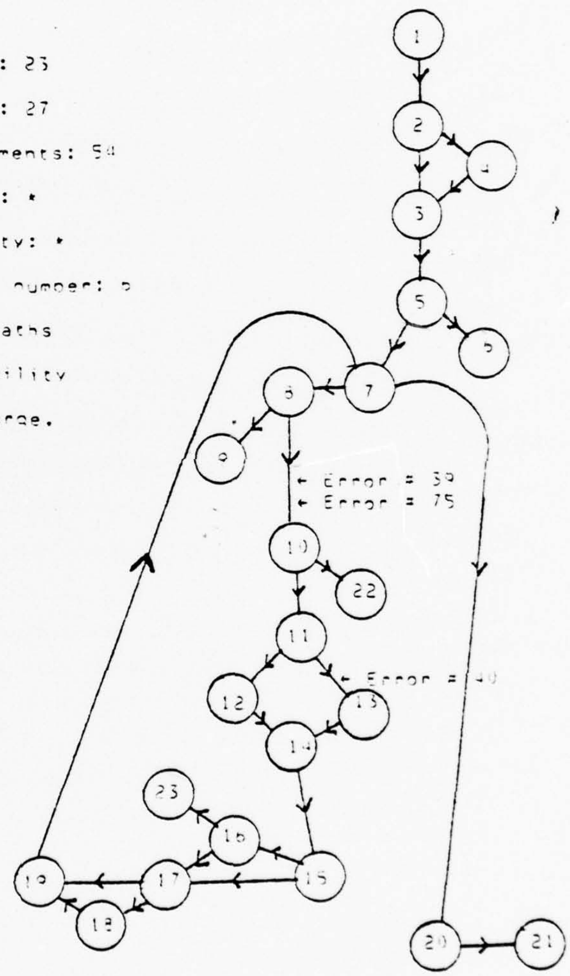
c) # of statements: 54

d) # of paths: *

e) Reachability: *

f) Cyclomatic number: 6

* Number of paths and reachability are very large.



DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : GET SUCCESSORS

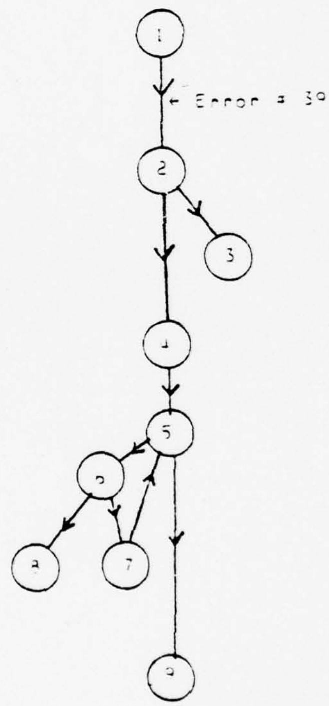
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 34
NUMBER OF NODES: 9
NUMBER OF ARCS : 9
NUMBER OF PATHS: 5
CYCLOMATIC NUMBER: V(G)= 2

REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	1
NODE	3 :	1
NODE	4 :	1
NODE	5 :	2
NODE	6 :	2
NODE	7 :	1
NODE	8 :	2
NODE	9 :	2
SUM:		13.00000

REACHABILITY INDEX
OF DIRECTED GRAPH: 1.444444



DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : OCCURS TWICE

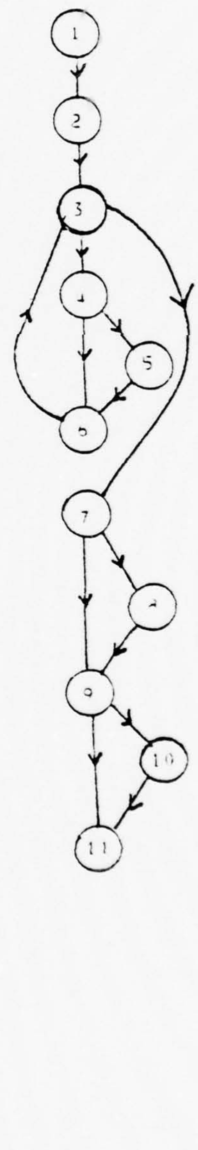
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 22
NUMBER OF NODES: 11
NUMBER OF ARCS : 14
NUMBER OF PATHS: 28
CYCLOMATIC NUMBER: V(G) = 5

REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	1
NODE	3 :	7
NODE	4 :	5
NODE	5 :	5
NODE	6 :	6
NODE	7 :	7
NODE	8 :	7
NODE	9 :	14
NODE	10 :	14
NODE	11 :	28
SUM:		93.00000

REACHABILITY INDEX
OF DIRECTED GRAPH: 8.454545



DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : ONE WAY

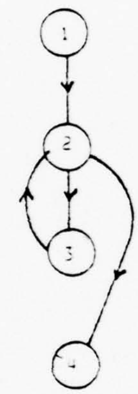
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 11
NUMBER OF NODES: 4
NUMBER OF ARCS: 4
NUMBER OF PATHS: 2
CYCLOMATIC NUMBER: V(G)= 2

REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	2
NODE	3 :	1
NODE	4 :	2
SUM:		6.000000

REACHABILITY INDEX
OF DIRECTED GRAPH: 1.500000



DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : MATCHING

a) # of nodes : 7

b) # of arcs : 9

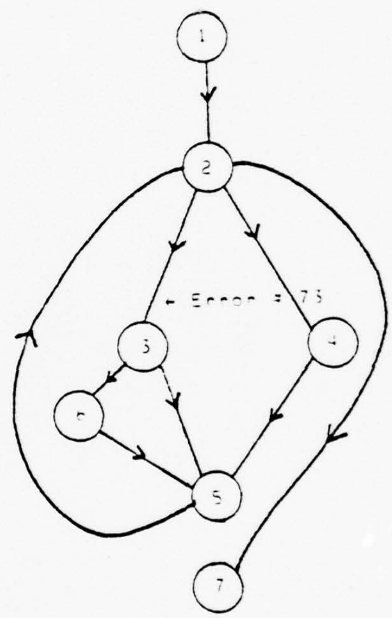
c) # of statements: 19

d) # of paths: *

e) Reachability: *

f) Cyclomatic number: 4

* Number of paths
and reachability
are very large.



DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : CHECKBACK

a) # of nodes: 13

b) # of arcs : 16

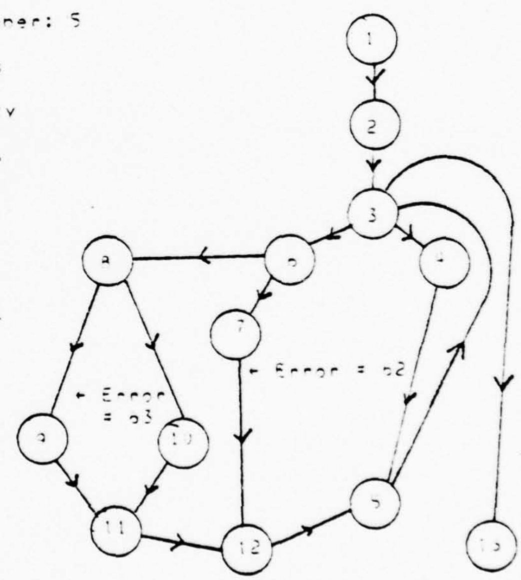
c) # of statements: 30

d) # of paths: *

e) Reachability: *

f) Cyclomatic number: 5

* Number of paths and reachability are very large.



DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : SAVE MEMORY

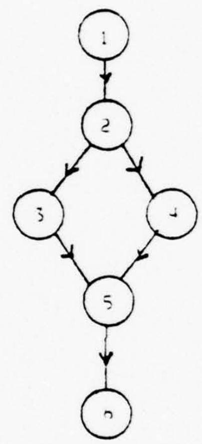
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 24
NUMBER OF NODES: 6
NUMBER OF ARCS : 6
NUMBER OF PATHS: 2
CYCLOMATIC NUMBER: V(G)= 2

REACHABILITY OF NODES:

NODE 1	:	1
NODE 2	:	1
NODE 3	:	1
NODE 4	:	1
NODE 5	:	2
NODE 6	:	2

SUM: 8.000000
REACHABILITY INDEX
OF DIRECTED GRAPH: 1.333333



DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : LIST PATH

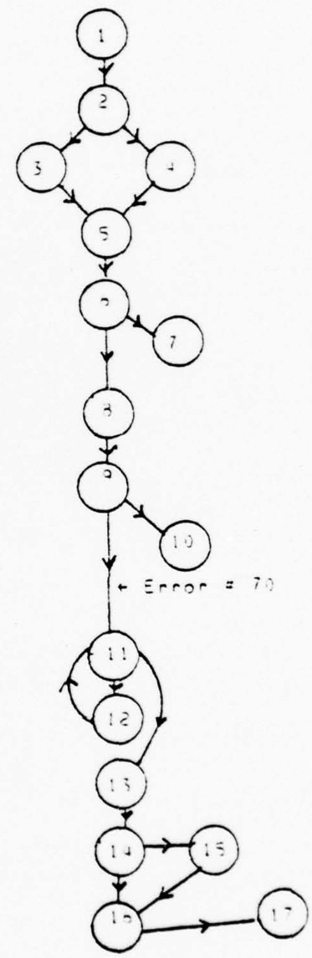
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 26
NUMBER OF NODES: 17
NUMBER OF ARCS : 19
NUMBER OF PATHS: 12
CYCLOMATIC NUMBER: V(G)= 4

REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	1
NODE	3 :	1
NODE	4 :	1
NODE	5 :	2
NODE	6 :	2
NODE	7 :	2
NODE	8 :	2
NODE	9 :	2
NODE	10 :	2
NODE	11 :	4
NODE	12 :	2
NODE	13 :	4
NODE	14 :	4
NODE	15 :	4
NODE	16 :	8
NODE	17 :	8
SUM:		50.00000

REACHABILITY INDEX
OF DIRECTED GRAPH: 2.941176



DIRECTED GRAPH REPRESENTATION

PROJECT = : 2

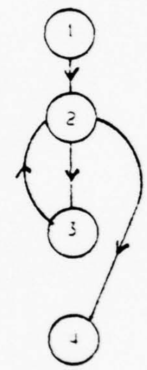
Program part : LIST ALL PATHS, MAIN

COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 10 - 11
NUMBER OF NODES: 4
NUMBER OF ARCS : 4
NUMBER OF PATHS: 2
CYCLOMATIC NUMBER: V(G)= 2

REACHABILITY OF NODES:
NODE 1 : 1
NODE 2 : 2
NODE 3 : 1
NODE 4 : 2

SUM: 6.000000
REACHABILITY INDEX
OF DIRECTED GRAPH: 1.500000



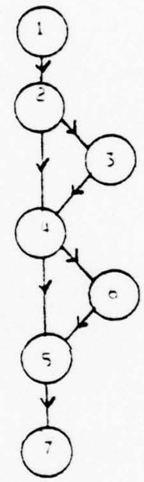
DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : ANALYZE

COMPLEXITY MEASURES:

```
NUMBER OF STATEMENTS: 23
NUMBER OF NODES:     7
NUMBER OF ARCS :     8
NUMBER OF PATHS:     4
CYCLOMATIC NUMBER:  V(G)=    3

REACHABILITY OF NODES:
NODE    1  :    1
NODE    2  :    1
NODE    3  :    1
NODE    4  :    2
NODE    5  :    4
NODE    6  :    2
NODE    7  :    4
               ---------
     SUM:       15.00000
REACHABILITY INDEX
OF DIRECTED GRAPH:          2.142857
```



DIRECTED GRAPH REPRESENTATION

PROJECT # : 2

Program part : SET PATHS

a) # of nodes: 40

b) # of arcs : 54

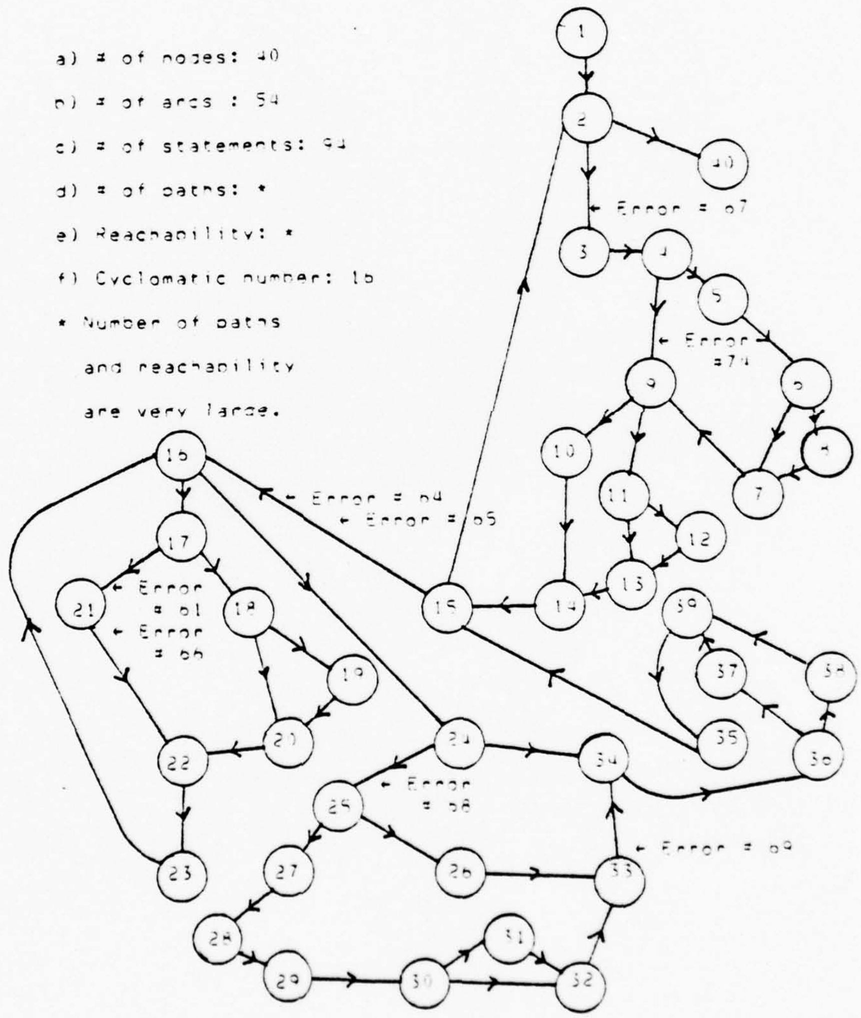
c) # of statements: 94

d) # of paths: *

e) Reachability: *

f) Cyclomatic number: 16

* Number of paths and reachability are very large.



TEST PHASE DESCRIPTION

Project # : 2

Test run # : 1   Including 1 Test Step

Begin of Test (day/time) : 03/17/1500   End of Test (day/time) : 03/17/1900

CPU time for necessary compiles (in sec.): 7.53
a) 7.53 b)   c)   d)   e)   f)   g)

CPU time for TEST run (sec) : 25.29

Man Hours for this Test run : 3.0 (including preparation of tests)

TEST STEP	OBJECTVE	EXPECTED RESULT (TOLERANCE)	ACTUAL RESULT	ERROR[1] #	DAY TIME	COMMENTS AND CODED ERROR TYPES
						1) Record when error occurs.
1	Test performance of program for various directed graphs:				3/17 1500	
	a) directed graph 9 nodes, 10 arcs	3 valid paths	O.K.			
	b) directed graph 6 nodes, 7 arcs	3 valid paths	O.K.			
	c) directed graph 6 nodes, 7 arcs	4 valid paths	O.K.			
	d) directed graph 22 nodes, 25 arcs	7 valid paths	O.K.			
	e) directed graph 22 nodes, 27 arcs	11 valid paths	O.K.			
	f) directed graph 1 node, 1 arc (self loop at node 1)	no valid path warning should be printed	O.K.			
	g) directed graph 2 nodes, 2 arcs	2 valid paths	O.K.			
	h) directed graph 2 nodes, 1 arc	1 valid path	O.K.		3/17 1900	

Remarks: Directed graphs submitted for testing include a variety of boundary conditions and special cases which are considered to be difficult to examine.

TEST PHASE DESCRIPTION

Project # : 2

Test run # : 1 Including 1 Test Steps

Begin of Test (day/time) : 03/15/1200 End of Test (day/time) : 03/17/2200

CPU time for necessary compiles (in sec.): 6.83
a) 6.83 b) c) d) e) f) g)

CPU time for TEST run (sec) : 844.44

Man Hours for this Test run : 10.0 (including preparation of tests)

TEST STEP	OBJECTVE	EXPECTED RESULT (TOLERANCE)	ACTUAL RESULT	ERROR # [1)]	DAY TIME	COMMENTS AND CODED ERROR TYPES
						1) Record when error occurs.
1	Find limits of the program using a directed graph having an infinite number of paths according to the definition being used.	Program should list all paths until freelist M2 is exhausted. Upon termination of the program an appropriate warning should be printed, such that the user can interpret the results of the path analysis.	O.K.		3/15 1200 – 3/17 2200	

Remarks: Due to extensive debugging previous to testing no errors have been found during the testing phase.
Test data for this run are chosen to indicate limitations of the program because of the finiteness of time and memory available.

APPENDIX C

PROJECT DESCRIPTION

Project # : 3

Project title: PATH ANALYSIS IN DIRECTED GRAPHS
WITH RESPECT TO REACHABILITY OF NODES

Programmer : HOFFMANN

Programming Language : ALGOL

Programming environment: IBM/360/67,OS/MVT,BATCH

Design notes : see ANNEX A

Program listing : see ANNEX B

Coding notes : see ANNEX C

Debugging notes : see ANNEX D

Error Listing : see ANNEX E

Final statistics : see ANNEX F

Graphical representation : see ANNEX G

Test phase description: see ANNEX H

Starting date: 1 MAR 77 Ending date: 17 MAR 77

EXPERIMENT DESCRIPTION

1. Project description:

This program is an extension of project # 2. The graph analyzing program as developed in project # 2 was modified to calculate the reachability of each node. A summary of the calculations is printed at the end of the analysis. All features described in project # 2 will be unchanged.

Input: via punch cards

Output: via line printer

(For input description and error messages see project description of project # 2.)

2. Programmer's background:

a) Experience in programming:

Oct 1970 - May 1971	Programming courses
May 1971 - April 1972	Module Programmer
May 1972 - June 1974	Work in Test and Simulation Department at the NAVAL COMMAND AND CONTROL SYSTEMS COMMAND (FEDERAL GERMAN NAVY) Testing of tactical real time systems
March 1975 - Jan 1977	Student at the NAVAL POSTGRADUATE SCHOOL, Monterey, Computer Science

b) Experience in testing:

Two years of work in testing and simulation.

c) Experience in the area of the given problem: None.

d) Experience in the progamming language being used:

Experience over a period of 18 months in more than 10 programming projects. (Total number of source statements produced during that time was more 4000.)

3. Psychological factors:

a) Did the programmer like the project? Yes.

b) How does the programmer like the programming

language?

Favorite programming language.

c) Was the programmer satisfied by the way the problem was specified?

Only minor criticism.

d) How did the programmer like the programming environment?

The facilities (study room, card punch room) were not conducive to efficient programming because of restricted space, bad lighting and noise.

e) Other factors:

The recording of the experiment's data during the project affected speed and concentration considerably.

4. Comments on Documentation

For the documentation of each software development phase a special documentation form has been developed. These forms are designed to provide a firm guideline for the experiment programmer to record all data of interest for subsequent error analysis.

- Begin and end of each step was recorded with respect to day and time.
- Each error was recorded when it is discovered. The error was then identified by a unique error number (1,2,...). Furthermore the time of discovery and the error type (using error types listed in ANNEX F) were recorded.
- If appropriate, comments about error discovery, reason why the error was made, etc. were documented in ANNEX E2.
- For each error the phase in which the error was made, the phase in which the error was discovered and the time spent to correct the error was recorded in ANNEX E1.
- For each step in any one of the software development phases the day/time of begin and end was recorded. In addition, the time (in man hours) for each step was recorded. This excludes the overhead used for documentation of the experiment data.

WORKSHEET FOR DESIGN PHASE AND DESIGN REVIEW PHASE OF PRCJECT # 3

STEP #	PROBLEM AND PLANNED SOLUTION	ALTERNATE SOLUTIONS	DAY TIME	MAN HOURS /STEP	ERROR #	COMMENT
1	Analyze ways to compute the reachability index for each node. -In general the reachability index of a node is incremented by 1 when the node is added to the current path or out as the first node on an alternative path. -The reachability of nodes which belong to a path which has to be removed has to be decremented by 1 up to the node which has been the last branch point on this path ex-cluding the branch point itself.		3/19 1500 3/19 1800	3.0		
2	Top-Down Design: a) Define additional subrou-tines: - Update Reachability (increment by 1) - Correct Reachability (decrement by 1 for all pre-vious nodes up to and exclu-ding the last branch point) - List Reachability (output of results) b) Define necessary changes within current version of the analyzing program.		3/20 1600 3/20 2000	4.0		

Annex B

Program Listing of Projects # 2 and 3

```
COMMENT              ANALYSIS OF DIRECTED GRAPHS;

BEGIN
COMMENT              DATA DEFINITIONS;

BITS ARRAY M1(0::451);        COMMENT USED TO IMPLEMENT
                                       LIST OF SUCCESSORS FOR
                                       EACH NODE IN THE NETWORK.;
BITS ARRAY M2(0::40002);      COMMENT USED TO IMPLEMENT
                                       NETWORK INFORMATION AND
                                       LIST OF POSSIBLE PATHS;
COMMENT              POINTERS;
INTEGER PATHHEAD;             COMMENT POINTS TO BEGIN OF LINKED LIST
                                       OF PATHS;
INTEGER LASTPATH;             COMMENT POINTS TO LAST OF THE LIST;
INTEGER CURRENTPATH;          COMMENT POINTS TO CURRENT PATH;
INTEGER CURRENTNODE;          COMMENT POINTS TO CURRENT NODE;
INTEGER LASTOCCURRENCE;       COMMENT POITS TO LAST KNOWN OCCURRENCE
                                       OF CURRENT NODE (SAME NAME);
INTEGER BRANCH_POINT;         COMMENT POINTS LAST BRANCH POINT ON
                                       THE CURRENT PATH;
INTEGER MAX1, MAX2;           COMMENT POINTERS TO LAST ITEMS OF M1, M2;

COMMENT              COUNTERS AND OTHER INTEGERS;

INTEGER NUMBER_OF_PATHS;      COMMENT INFORMATION ABOUT TOTAL NUMBER
                                       OF PATHES;
INTEGER NUMBER_OF_NODES;      COMMENT TOTAL NUMBER OF ORIGINAL NODES;
INTEGER NUMBER_OF_ARCS;       COMMENT TOTAL NUMBER OF ARCS IN THE GRAPH;
INTEGER PATHS_PRINTED;        COMMENT COUNTER OF PATHS BEING PRINTED;
INTEGER DISTANCE;             COMMENT DISTANCE ON CURRENT PATH BETWEEN
                                       CURRENT NODE AND A PREVIOUS
                                       OCCURRENCE ON THE PATH(SAME NAME);
INTEGER CURRENT_PATH_ID;      COMMENT PATH IDENTIFICATION OF CURRENT
                                       PATH;
INTEGER MAINSWITCH;           COMMENT INDICATES TERMINATION/CONTINUATION;
INTEGER TOGGLE;
INTEGER CN;
INTEGER I;

COMMENT              PARAMETERS;

INTEGER NIL;                  COMMENT INDICATES THE END OF A LINKED LIST;
INTEGER N1;                   COMMENT NUMBER OF ITEMS OF M1;
INTEGER N2;                   COMMENT NUMBER OF ITEMS OF M2;
INTEGER K1;                   COMMENT SIZE OF ITEMS OF M1;
INTEGER K2;                   COMMENT SIZE OF ITEMS OF M2;
INTEGER MAXSIZE;              COMMENT MAXIMUM SIZE OF NODE NAME;
```

```
COMMENT              PRIMITIVES;

PROCEDURE INITIALIZE1;
    BEGIN
    INTEGER I;
    MAX1:=(N1-1)*K1;
    FOR I:= 0 STEP K1 UNTIL MAX1 DO SETCDR1(I,I+K1);
    SETCDR1(MAX1,NIL);
    END INITIALIZE1;

PROCEDURE INITIALIZE2;
    BEGIN
    INTEGER I;
    MAX2:=(N2-1)*K2;
    FOR I:= 0 STEP K2 UNTIL MAX2 DO SETCDR2(I,I+K2);
    SETCDR2(MAX2,NIL);
    M2(1):=#CCC00CCC0;    COMMENT SAFETY MEASURE;
    SETCAR2(0,0);         COMMENT SAFETY MEASURE;
    END INITIALIZE2;

INTEGER PROCEDURE CAR1(INTEGER VALUE X);
COMMENT RETURNS VALUE OF CARFIELD OF ELEMENT X IN M1;
    NUMBER(M1(X) SHR 16);

INTEGER PROCEDURE CAR2(INTEGER VALUE X);
COMMENT RETURNS VALUE OF CARFIELD OF ELEMENT X IN M2;
    NUMBER(M2(X) SHR 16);

INTEGER PROCEDURE CDR1(INTEGER VALUE X);
COMMENT RETURNS VALUE OF CDRFIELD IN M1;
    NUMBER(M1(X) AND #FFFF);

INTEGER PROCEDURE CDR2(INTEGER VALUE X);
COMMENT RETURNS VALUE OF CDRFIELD IN M2;
    NUMBER(M2(X) AND #FFFF);

PROCEDURE SETCAR1(INTEGER VALUE X,Y);
COMMENT SET CARFIELD OF ELEMENT X IN M1 TO Y;
    M1(X):=(M1(X) AND #FFFF) OR (BITSTRING(Y) SHL 16);

PROCEDURE SETCAR2(INTEGER VALUE X,Y);
COMMENT SET CARFIELD OF ELEMENT X IN M2 TO Y;
    M2(X):=(M2(X) AND #FFFF) OR (BITSTRING(Y) SHL 16);

PROCEDURE SETCDR1(INTEGER VALUE X,Y);
COMMENT SET CDRFIELD OF ELEMENT X IN M1 TO Y;
```

```
   M1(X):=(M1(X) AND #FFFF0C000) OR BITSTRING(Y);

PROCEDURE SETCDR2(INTEGER VALUE X,Y);
COMMENT SET CDRFIELD OF ELEMENT X IN M2 TO Y;
   M2(X):=(M2(X) AND #FFFF0C000) OR BITSTRING(Y);

INTEGER PROCEDURE ALLOCATE1;
COMMENT ALLOCATE NEXT ITEM OF M1;
   BEGIN
   INTEGER A;
   A:=CDR1(0);      COMMENT FREELIST BEGINS AT ZERO;
   IF A = NIL THEN ERROR(1,0);
   SETCDR1(0,CDR1(A));
   M1(A):=#00C00000;            COMMENT CLEAR ITEM;
   A
   END ALLOCATE1;

INTEGER PROCEDURE ALLOCATE2;
COMMENT ALLOCATE NEXT ITEM OF M2;
   BEGIN
   INTEGER A;
   A:=CDR2(0);      COMMENT FREELIST BEGINS AT ZERO;
   IF A = NIL THEN ERROR(2,0);
   SETCDR2(0,CDR2(A));
   M2(A):=#C00000C0;            COMMENT CLEAR ITEM;
   M2(A+1):=#C000C000;
   A
   END ALLOCATE2;

PROCEDURE FREE1(INTEGER VALUE X);
COMMENT RETURN ITEM TO FREELIST M1;
   BEGIN
   SETCDR1(X,CDR1(0));
   SETCDR1(0,X);
   END FREE1;

PROCEDURE FREE2(INTEGER VALUE X);
COMMENT RETURN ITEM TO FREELIST M2;
   BEGIN
   SETCDR2(X,CDR2(0));
   SETCDR2(0,X);
   END FREE2;


COMMENT           UTILITIES;

PROCEDURE SPACE(INTEGER VALUE N);
COMMENT WRITE N BLANK LINES;
```

```
    BEGIN
    INTEGER I;
    FOR I:=1 STEP 1 UNTIL N DO WRITE(" ");
    END SPACE;

PROCEDURE INITIALIZE_ALL;
COMMENT INITIAL SET UP;
    BEGIN
    COMMENT SET ALL PARAMETERS;
    NIL := 0;
    N1 := 451;
    N2 := 20001;
    K1 := 1;
    K2 := 2;
COMMENT REDUCE FREELISTS FOR DEBUGGING PURPOSES;
    INTFIELDSIZE := 5;
    PATHS_PRINTED:=0;
    NUMBER_OF_NODES:=0;
    COMMENT INITIALIZE LINKED LISTS;
    INITIALIZE1;
    INITIALIZE2;
    END INITIALIZE_ALL;

COMMENT                ERROR HANDLING;

PROCEDURE ERROR(INTEGER VALUE N,INFO);
COMMENT HANDLES ALL ERRORS AND EXEPTIONAL CASES;
    BEGIN
    INTFIELDSIZE := 5;
    WRITE("ERROR : ",N);
    CASE N OF
        BEGIN
        COMMENT ERROR 1;
            BEGIN
            WRITE("FREELIST 1 EXHAUSTED, EXECUTION ENDS.");
            DIAGNOSE(FALSE,TRUE);
            GO TO OVERFLOW;
            END;
        COMMENT ERROR 2;
            BEGIN
            WRITE("FREELIST 2 EXHAUSTED, ANALYSIS TERMINATED.");
            DIAGNOSE(TRUE,FALSE);
            END;
        COMMENT ERROR 3;
            BEGIN
            WRITE("SON",INFO," OF CURRENT NODE:",CURRENTNODE,
                  " NOT FOUND");
            DIAGNOSE(TRUE,TRUE);
```

```
    END;
COMMENT ERROR 4;
    BEGIN
    WRITE("DUPLICATION OF CURRENT PATH:",CURRENT_PATH_ID,
          " IMPOSSIBLE");
    DIAGNOSE(TRUE,TRUE);
    END;
COMMENT ERROR 5;
    BEGIN
    WRITE("INVALID PATH ID:",INFO);
    DIAGNOSE(TRUE,TRUE);
    END;
COMMENT ERROR 6;
    BEGIN
    WRITE("PATH",INFO,"NOT FOUND");
    DIAGNOSE(TRUE,TRUE);
    END;
COMMENT ERROR 7;
    BEGIN
    WRITE("CANNOT FIND END OF INVALID PATH ID:",INFO);
    DIAGNOSE(TRUE,TRUE);
    END;
COMMENT ERROR 8;
    BEGIN
    WRITE("INPUT VIOLATES SPECIFICATIONS:");
    WRITE("FIRST NODE ID:",INFO);
    WRITE("JOB TERMINATED.");
    GO TO OVERFLOW;
    END;
COMMENT ERROR 9;
    BEGIN
    WRITE("INVALID NUMBER OF SUCCESSORS:",INFO);
    WRITE("JOB TERMINATED.");
    GO TO OVERFLOW;
    END;
COMMENT ERROR 10;
    BEGIN
    WRITE("CANNOT FIND SUCCESSOR WITH ID:",INFO);
    DIAGNOSE(TRUE,TRUE);
    END;
COMMENT ERROR 11;
    BEGIN
    WRITE("CANNOT LIST PATH BEGINNING AT NIL.");
    DIAGNOSE(TRUE,TRUE);
    END;
COMMENT ERROR 12;
    BEGIN
    WRITE("PATH HEADER POINTS TO NIL. NO PATH FOUND.");
```

```
            DIAGNOSE(TRUE,TRUE);
            END;
        COMMENT ERROR 13;
            BEGIN
            WRITE("WARNING:");
            WRITE("ALL PATHS HAVE BEEN REMOVED FROM THE SYSTEM.");
            WRITE("PATHHEAD POINTS TO NIL. CANNOT CONTINUE.");
            END;
        END;
    END ERROR;

PROCEDURE DIAGNOSE(LOGICAL VALUE TERMINATION,SELECT);
COMMENT DEBUGGING AND TESTING TOOL;
    BEGIN
    INTEGER I;
    SPACE(2);
    WRITE("DIAGNOSTICS:");
    SPACE(1);
    WRITE("CURRENTNODE:",CURRENTNODE,",LASTOCCURRENCE:",LASTOCCURRENCE);
    WRITE("DISTANCE:", DISTANCE);
    WRITE("NUMBER_OF_PATHS:",NUMBER_OF_PATHS);
    WRITE("PATHHEAD:",PATHHEAD);
    WRITE("CURRENT_PATH_ID:",CURRENT_PATH_ID);
    WRITE("CURRENTPATH:",CURRENTPATH,",LASTPATH:",LASTPATH);
    WRITE("BRANCH_POINT:",BRANCH_POINT);
    WRITE("NUMBER_OF_NODES:",NUMBER_OF_NODES);
    WRITE("NUMBER_OF_ARCS:",NUMBER_OF_ARCS);
    WRITE("PATHS_PRINTED:",PATHS_PRINTED);
    SPACE(1);
    WRITE("COMPLEXITY MEASURES AT POINT OF TERMINATION:");
    LIST_REACHABILITY;
    WRITE("WARNING: NUMBER OF PATHS AND REACHABILITY IS INCOMPLETE.");
    SPACE(2);
    IF SELECT = TRUE THEN
        BEGIN
        WRITE("ELEMENT    CAR1     CDR1");
        FOR I:=CDR1(0) STEP -K1 UNTIL 0 DO  WRITE(I,CAR1(I),CDR1(I));
        WRITE("ELEMENT    CAR2     CDR2");
        FOR I:=CDR2(0)+1 STEP -K1 UNTIL 0 DO WRITE(I,CAR2(I),CDR2(I));
        END;
    WRITE("END OF DIAGNOSTICS");
    IF TERMINATION = TRUE THEN
        BEGIN
        WRITE("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++");
        WRITEON("++++++++++++++++++++++++++++++++++");
        GO TO TERMINATE;
        END;
    END DIAGNOSE;
```

```
COMMENT              UTILITIES FOR M1;

PROCEDURE SETNAME(INTEGER VALUE X,NAME);
COMMENT SET NAMEFIELD OF ITEM X TO NAME;
    SETCAR1(X,NAME);

PROCEDURE ADDBROTHER(INTEGER VALUE PX,PY);
COMMENT LINK BROTHER POINTED AT BY PX WITH BROTHER POITED AT BY PY;
    SETCDR1(PX,PY);

INTEGER PROCEDURE NAME(INTEGER VALUE X);
COMMENT RETURNS NAME OF ELEMENT X IN M1;
    CAR1(X);

INTEGER PROCEDURE BROTHER(INTEGER VALUE X);
COMMENT RETURNS POINTER TO BROTHER OF X;
    CDR1(X);

INTEGER PROCEDURE SON(INTEGER VALUE I,PARENT);
COMMENT RETURNS NAME OF ITH SON OF PARENT NODE. THE PARENT NODE IS
          IDENTIFIED BY THE ORIGIN POINTER WITHIN THE ITEM BEING POINTED
          AT BY PARENT;
    BEGIN
    INTEGER NEXT_SON,J,SONS_NAME;
    J:=1;
    IF NUMBER_OF_SUCCESSORS(PARENT) = 0 THEN SONS_NAME:=NIL
    ELSE
        BEGIN
        IF I > NUMBER_OF_SUCCESSORS(PARENT) THEN ERROR(3,I);
        COMMENT GET POINTER TO FIRST SON;
        NEXT_SON:=CDR1(CDR2(CAR2(PARENT+1)));
        COMMENT FIND ITH SON;
        WHILE J <= I DO
            BEGIN
            SONS_NAME:=NAME(NEXT_SON);
            NEXT_SON:=BROTHER(NEXT_SON);
            J:=J+1;
            END;
        END;
    SONS_NAME
    END SON;
```

```
COMMENT            UTILITIES M2;

INTEGER PROCEDURE NAME_OF(INTEGER VALUE POINTER);
COMMENT RETURNS NAME OF ITEM  BEING POINTED AT BY POINTER;
   CDR2(POINTER+1);

INTEGER PROCEDURE PATH_ID(INTEGER VALUE POINTER);
COMMENT RETURNS PATHNUMBER OF PATH LIST ELEMENT BEING POINTED AT;
   CDR2(POINTER+1);

INTEGER PROCEDURE NUMBER_OF_SUCCESSORS(INTEGER VALUE POINTER);
COMMENT RETURNS NUMBER OF SUCCESSORS OF NODE BEING POINTED AT BY
        EXTRACTING THE APPROPRIATE VALUE FROM THE CORRESPONDING
        ITEM REFFERRED TO BY THE NAME OF THE ELEMENT BEING POINTED AT;
   CAR2(CAR2(POINTER+1));

INTEGER PROCEDURE PREDECESSOR(INTEGER VALUE POINTER);
COMMENT RETURNS THE POINTER TO THE PREDECESSOR WITHIN THE SAME PATH;
   CDR2(POINTER);

INTEGER PROCEDURE SUCCESSOR(INTEGER VALUE POINTER);
COMMENT RETURNS THE POINTER TO THE SUCCESSOR WITHIN THE SAME PATH;
   CAR2(POINTER);

PROCEDURE INITPATHLIST;
COMMENT INITIAL SET UP OF LINKED LISTS OF POSSIBLE PATHS;
   BEGIN
   INTEGER A;
   NUMBER_OF_PATHS:=1;
   CURRENT_PATH_ID:=1;
   PATHHEAD:=ALLOCATE2;
   CURRENTPATH:=PATHHEAD;
   LASTPATH:=PATHHEAD;
   SET_PATH_ID(PATHHEAD,1);
   A:=DUPLICATE1(2); COMMENT DUPLICATE NODE 1 (LOCATED AT M2(2));
   LINKPATH(CURRENTPATH,NIL);
   LINKFORWARD(CURRENTPATH,A);
   CURRENTNODE:=A;
   UPDATE_REACHABILITY(CURRENTNODE,1);  COMMENT INITIALIZE REACHABILITY
                                                 OF NODE 1;
   LASTOCCURRENCE:=A;
   DISTANCE:=0;
   END INITPATHLIST;

PROCEDURE SET_PATH_ID(INTEGER VALUE ITEM,ID);
COMMENT SET THE PATH IDENTIFICATION OF ITEM;
   SETCDR2((ITEM+1),ID);
```

```
PROCEDURE PUT_NEXT_PATHHEADER;
COMMENT ADD THE HEADER OF A NEW PATH TO THE LINKED LIST OF PATHS.
        UPDATE NUMBER_OF_PATHS, LASTPATH AND SET APPROPRIATE PATH_ID;
   BEGIN
   INTEGER A;
   A:=ALLOCATE2;
   LINKPATH(LASTPATH,A);
   LINKPATH_BACK(A,LASTPATH);
   LINKPATH(A,NIL);
   NUMBER_OF_PATHS:=NUMBER_OF_PATHS+1;
   SET_PATH_ID(A,NUMBER_OF_PATHS);
   LASTPATH:=A;
   END PUT_NEXT_PATHHEADER;

INTEGER PROCEDURE NEXT_PATH(INTEGER VALUE POINTER);
COMMENT RETURN POINTER TO PATHHEADER OF NEXT PATH;
   CDR2(POINTER);

INTEGER PROCEDURE PREVIOUS_PATH(INTEGER VALUE POINTER);
COMMENT RETURN POINTER TO PATHHEADER OF PREVIOUS PATH;
   CAR2(POINTER+1);

PROCEDURE LINKPATH_BACK(INTEGER VALUE X,Y);
COMMENT LINK BACKWARD FROM PATHHEADER X TO PATHHEADER Y;
   SETCAR2((X+1),Y);

PROCEDURE LINKPATH(INTEGER VALUE X,Y);
COMMENT LINKFORWARD FROM PATHHEADER X TO PATHHEADER Y.
        X AND Y ARE POINTERS;
   SETCDR2(X,Y);

PROCEDURE LINKFORWARD(INTEGER VALUE X,Y);
COMMENT LINK NODE X WITH ITS SUCCESSOR NODE Y;
   SETCAR2(X,Y);

PROCEDURE LINKBACK(INTEGER VALUE X,Y);
COMMENT LINK NODE X WITH ITS PREDECESSOR  NODE Y;
   SETCDR2(X,Y);

INTEGER PROCEDURE DUPLICATE1(INTEGER VALUE X);
COMMENT DUPLICATE ITEM BEING POITED AT BY X. SET FORWARD POITER TO 1
        AND BACKWARD POINTER TO NIL. DUPLICATES ONLY ITEMS OF M2;
   BEGIN
   INTEGER A;
   A:=ALLOCATE2;
   M2(A):=#10000;
   M2(A+1):=M2(X+1);
   A
```

```
    END DUPLICATE1;

PROCEDURE ADDNODE(INTEGER VALUE RESULT POINTER; INTEGER VALUE ORIGINAL);
COMMENT ADDS A NODE WITH SAME NAME TO THE NODE BEING POITED AT.THIS
          INCLUDES LINKING IN BOTH DIRECTIONS AND UPDATING OF POINTER.
          THE RETURN VALUE OF POINTER WILL BE POITING TO THE ADDED NODE;
    BEGIN
    INTEGER D;
    D:=DUPLICATE1(ORIGINAL);      COMMENT REFERS TO ORIGINAL INPUT;
    LINKFORWARD(POINTER,D);
    LINKBACK(D,POINTER);
    POINTER:=D;
    END ADDNODE;

COMMENT   -         PROCEDURAL SUBROUTINES;

INTEGER PROCEDURE FIND_ORIGINAL(INTEGER VALUE ID);
COMMENT RETURN POINTER TO ORIGINAL INPUT OF A NODE IDENTIFIED BY ID;
    BEGIN
    INTEGER I,POINTER;
    I:=2;
    POINTER:=NIL;
    WHILE ((I<=NUMBER_OF_NODES*2) AND (POINTER = NIL)) DO
         BEGIN
         IF NAME_OF(I) = ID THEN POINTER:=I;
         I:=I+K2;
         END;
    IF POINTER = NIL THEN ERROR(10,ID);
    POINTER
    END FIND_ORIGINAL;

INTEGER PROCEDURE DUPLICATE_CURRENT_PATH;
COMMENT CURRENT PATH WILL DUPLICATED EXCLUDING THE CURRENT NODE AND
          ADDED TO THE LINKED LIST OF PATHS. THOS INCLUDES AN UPDATE
          OF NUMBER_OF_PATHS AND POINTER TO LASTPATH("LASTPATH").
          A POINTER TO THE LAST NODE OF THE DUPLICATED PATH WILL BE
          RETURNED;
    BEGIN
    INTEGER X,Y;
    IF PREDECESSOR(CURRENTNODE) = NIL THEN ERROR(4,0)
    ELSE
         BEGIN
         PUT_NEXT_PATHHEADER;
         X:=SUCCESSOR(CURRENTPATH);
         Y:=LASTPATH;
         WHILE X ¬= CURRENTNODE DO
              BEGIN
              ADDNODE(Y,CAR2(X+1));
```

```
            IF PREDECESSOR(X) = NIL THEN LINKBACK(Y,NIL);
            X:=SUCCESSOR(X);
            END;
        END;

    END DUPLICATE_CURRENT_PATH;

PROCEDURE ALTERNATIVE_WAY(INTEGER VALUE NEXT_SONS_NAME);
COMMENT PRODUCES A DUPLICATE OF CURRENT PATH WITH DIFFERENT CURRENT NODE
        AND LINKS IT TO THE END OF THE LINKED LIST OF POSSIBLE PATHES;
    BEGIN
    INTEGER POINTER;
    INTEGER ORIGIN;
    ORIGIN:=FIND_ORIGINAL(NEXT_SONS_NAME);
    POINTER:=DUPLICATE_CURRENT_PATH;
    ADDNODE(POINTER,ORIGIN);
    UPDATE_REACHABILITY(ORIGIN,1);
    END ALTERNATIVE_WAY;

INTEGER PROCEDURE FIND_PATH(INTEGER VALUE ID);
COMMENT RETURN POINTER TO PATHHEADER OF PATH IDENTIFIED BY ID;
    BEGIN
    INTEGER NEXT;
    IF ID <= 0 THEN ERROR(5,ID);
    NEXT:=PATHHEAD;
    WHILE PATH_ID(NEXT) ¬= ID DO
        BEGIN
        IF NEXT = NIL THEN ERROR(6,NEXT);
        NEXT:=NEXT_PATH(NEXT);
        END;
    CURRENTPATH:=NEXT;
    CURRENT_PATH_ID:=PATH_ID(NEXT);
    NEXT
    END FIND_PATH;

PROCEDURE REMOVE_PATH(INTEGER VALUE RESULT POINTER);
COMMENT REMOVE PATH POINTED AT BY POINTER. RESET SUBSEQUENT PATHIDS.
        RETURN VALUE OF POINTER POINTS TO NEXT PATH. RESET PATHHEAD
        OR LASTPATH IF APPROPRIATE;
    BEGIN
    INTEGER NEXT,I;
    IF ((POINTER=LASTPATH) AND (POINTER=PATHHEAD)) THEN
        WRITE("REMOVAL OF LAST PATH.");
    IF POINTER = CURRENTPATH THEN
        BEGIN
        CURRENTNODE:=NIL;
        LASTOCCURRENCE:=NIL;
        DISTANCE:=0;
```

```
      END;
IF POINTER = PATHHEAD THEN
      BEGIN
      PATHHEAD:=NEXT_PATH(POINTER);
      LINKPATH_BACK(NEXT_PATH(POINTER),NIL);
      IF PATHHEAD = NIL THEN ERROR(13,0);
      END
ELSE
      BEGIN
      LINKPATH(PREVIOUS_PATH(POINTER),NEXT_PATH(POINTER));
      IF POINTER ¬= LASTPATH THEN
            BEGIN
            LINKPATH_BACK(NEXT_PATH(POINTER),PREVIOUS_PATH(POINTER));
            END
      ELSE
            BEGIN
            LASTPATH:=PREVIOUS_PATH(POINTER);
            END;
      END;
COMMENT RETURN NODES OF UNLINKED PATH;
NEXT:=POINTER;
I:=PATH_ID(POINTER);  COMMENT SAVE ID OF UNLINKED PATH;
COMMENT SPECIAL CASE: LAST PATH BUT ONE;
IF NEXT_PATH(POINTER) = LASTPATH THEN SET_PATH_ID(LASTPATH,I);
IF NEXT_PATH(POINTER) ¬= NIL THEN POINTER:=NEXT_PATH(POINTER)
ELSE POINTER:=PREVIOUS_PATH(POINTER);  COMMENT SAVE RETURN VALUE;
FREE2(NEXT);
WHILE (( SUCCESSOR(NEXT) ¬= NIL) AND (SUCCESSOR(NEXT) ¬= 1)) DO
      BEGIN
      FREE2(SUCCESSOR(NEXT));
      NEXT:=SUCCESSOR(NEXT);
      END;
CURRENT_PATH_ID:=PATH_ID(POINTER);
CURRENTPATH:=POINTER;
COMMENT UPDATE PATH IDS;
NEXT:=POINTER;
IF POINTER ¬= LASTPATH THEN
      BEGIN
      WHILE NEXT ¬= NIL DO
            BEGIN
            SET_PATH_ID(NEXT,I);
            NEXT:=NEXT_PATH(NEXT);
            I:=I+1;
            END;
      END;
NUMBER_OF_PATHS:=NUMBER_OF_PATHS - 1;
END REMOVE_PATH;
```

```
INTEGER PROCEDURE END_OF_PATH(INTEGER VALUE ID);
COMMENT FIND END OF PATH IDENTIFIED BY ID. UPDATE CURRENTNODE,
        LASTOCCURRENCE, DISTANCE AND CURRENTPATH;
    BEGIN
    INTEGER POINTER;
    IF ((ID<=0) OR (ID > NUMBER_OF_PATHS)) THEN ERROR(7,ID);
    IF ID  = CURRENT_PATH_ID THEN POINTER:=CURRENTPATH
    ELSE POINTER:= FIND_PATH(ID);
    CURRENTPATH:=POINTER;  COMMENT UPDATE CURRENTPATH;
    IF ID ¬=  PATH_ID(POINTER) THEN
        BEGIN
        WRITE("WARNING:");
        WRITE("CURRENT PATH AND CURRENT PATH ID DO NOT MATCH.");
        WRITE("EXECUTION CONTINUES:");
        END;
    WHILE SUCCESSOR(POINTER) ¬= 1   DO POINTER:=SUCCESSOR(POINTER);
    CURRENTNODE:=POINTER;
    LASTOCCURRENCE:=POINTER;
    DISTANCE:=0;
    POINTER
    END END_OF_PATH;

PROCEDURE INPUT_DIRECTED_GRAPH_INFORMATION;
COMMENT READ INFORMATION FROM CARDS. VALID NODE IDS ARE INTEGERS
        BETWEEN 1 AND 999;
    BEGIN
    INTEGER N, NODE_NAME, A;
    WRITE("BEGIN OF INPUT:");
    READ(TOGGLE);
    READ(MAXSIZE);
    IF ((MAXSIZE < 1) OR (MAXSIZE > 3)) THEN
        BEGIN
        WRITE("INVALID SIZE. EXECUTUION CONTINUES. ");
        WRITE("DEFAULT SIZE= 3");
        MAXSIZE:=3;
        END;
    WRITE(" ");
    READ(NODE_NAME);
    WRITE("NODE_NAME:",NODE_NAME);
    NUMBER_OF_ARCS:=0;
    IF NODE_NAME ¬= 1 THEN ERROR(8,NODE_NAME);
    WHILE NODE_NAME ¬= 99999 DO
        BEGIN
        IF ((NODE_NAME < 1) OR (NODE_NAME > 999)) THEN
            BEGIN
            WRITE("INVALID NODE NAME. ");
            WRITE("NODE NAME MUST BE A POSITIVE INTEGER AND LESS THAN",
```

```
                                                                        " 1000");
          WRITE("EXECUTION TERMINATED.");
          GO TO OVERFLOW;
          END;
      A:=ALLOCATE2;
      SETCAR2(A+1,A); COMMENT SET POINTER TO ORIGINAL;
      SETCDR2(A+1,NODE_NAME);  COMMENT SET NODE IDENTIFICATION;
      NUMBER_OF_NODES:=NUMBER_OF_NODES + 1;
      READON(N);  COMMENT READ NUMBER OF SUCCESSORS;
      NUMBER_OF_ARCS:=NUMBER_OF_ARCS + N;
      WRITEON("NUMBER OF SUCCESSORS:",N);
      IF ((N < 0) OR (N> 16)) THEN  ERROR(9,N);
      IF N > 0 THEN
          BEGIN
          SETCAR2(A,N);  COMMENT SET NUMBER OF SUCCESSORS;
          GET_SUCCESSORS(A,N);  COMMENT CREATE LINKED LIST OF SONS;
          END
      ELSE
          BEGIN
          COMMENT LINK TO REACHABILITY COUNTER;
          SETCDR2(A,ALLOCATE1);
          END;
      SPACE(1);
      READ(NODE_NAME);
      IF ((NODE_NAME < 1) OR (NODE_NAME > 999)) THEN
          BEGIN
          IF NODE_NAME ¬= 99999 THEN
              BEGIN
              WRITE("INVALID NODE NAME. ");
              WRITE("NODE NAME MUST BE A POSITIVE INTEGER AND ",
                                                  "LESS THAN 1000");
              WRITE("EXECUTION TERMINATED.");
              GO TO OVERFLOW;
              END;
          END;
      IF NODE_NAME ¬= 99999 THEN WRITE("NODE_NAME:", NODE_NAME);
    END;
  SPACE(3);
  WRITE("END OF INPUT:",NUMBER_OF_NODES," NODES HAVE BEEN READ.");
  WRITE("NUMBER OF ARCS:",NUMBER_OF_ARCS);
  SPACE(2);
  WRITE("CYCLOMATIC NUMBER:");
  WRITE("    V(G) = ",NUMBER_OF_ARCS - NUMBER_OF_NODES + 2);
  WRITE("************************************************************");
  SPACE(2);
  WRITE("PATH ANALYSIS BEGINS:");
  SPACE(1);
  END INPUT_DIRECTED_GRAPH_INFORMATION;
```

```
PROCEDURE GET_SUCCESSORS(INTEGER VALUE ORIGINAL,N);
COMMENT LINK LIST OF N SUCCESSORS TO THE ORIGINAL INPUT NODE;
   BEGIN
   INTEGER I, SONS_NAME, X, Y;
   I:=1;
   Y:=ALLOCATE1;
   SETCDR2(ORIGINAL,Y);  COMMENT X POINTS BEGIN OF SUCCESSOR LIST;
   X:=ALLOCATE1;
   SETCDR1(Y,X);    COMMENT X POINTS TO FIRST SON;
   READON(SONS_NAME);
   IF ((SONS_NAME < 1) OR (SONS_NAME > 999)) THEN
       BEGIN
       WRITE("INVALID NODE NAME. ");
       WRITE("NODE NAME MUST BE A POSITIVE INTEGER AND LESS THAN",
                                                           " 1000");
       WRITE("EXECUTION TERMINATED.");
       GO TO OVERFLOW;
       END;
   WRITE("SUCCESSOR",I,":",SONS_NAME);
   SETNAME(X,SONS_NAME);
   WHILE I < N DO
       BEGIN
       Y:=ALLOCATE1;
       READON(SONS_NAME);
       IF ((SONS_NAME < 1) OR (SONS_NAME > 999)) THEN
           BEGIN
           WRITE("INVALID NODE NAME. ");
           WRITE("NODE NAME MUST BE A POSITIVE INTEGER AND LESS THAN",
                                                               " 1000");
           WRITE("EXECUTION TERMINATED.");
           GO TO OVERFLOW;
           END;
       ADDBROTHER(X,Y);
       SETNAME(Y,SONS_NAME);
       I:=I+1;
       WRITE("SUCCESSOR",I,":",SONS_NAME);
       X:=Y;
       END;
   ADDBROTHER(X,NIL);
   END GET_SUCCESSORS;

COMMENT            LOGICAL SUBROUTINES;

LOGICAL PROCEDURE OCCURS_TWICE(INTEGER VALUE RESULT LAST;
                               INTEGER VALUE ID;
                               LOGICAL VALUE RESULT TOP);
COMMENT LOOKING BACKWARD ON THE SAME PATH BEGINNING AT "LAST" THE
```

```
        LOGICAL VALUE OF A PREVIOUS OCCURRENCE OF "ID" WILL BE
        RETURNED. DISTANCE WILL BE UPDATED AND THE RETURN VALUE OF
        LAST POINTS TO THE PREVIOUS OCCURRENCE(IF ANY). THE RETURN
        VALUE OF TOP WILL BE SET TRUE IF ALL NODES ON THE PATH HAVE
        BEEN COMPARED AND "ID" HAS NOT BEEN FOUND;
   BEGIN
   LOGICAL TWICE;
   INTEGER POINTER;
   TWICE:=FALSE;
   POINTER:=LAST;
   WHILE ((TWICE = FALSE) AND (PREDECESSOR(POINTER) ¬= NIL)) DO
       BEGIN
       DISTANCE:=DISTANCE + 1;
       POINTER:=PREDECESSOR(POINTER);
       IF NAME_OF(POINTER) = ID THEN
           BEGIN
           TWICE:=TRUE;
           LAST:=POINTER;
           END;
       END;
   IF ((PREDECESSOR(POINTER) = NIL) AND (TWICE = FALSE)) THEN TOP:=TRUE;
   IF TWICE = FALSE THEN
       BEGIN
       LAST:=CURRENTNODE;
       DISTANCE:=0;
       END;
   TWICE
   END OCCURS_TWICE;

LOGICAL PROCEDURE ONE_WAY;
COMMENT RETURN LOGICAL VALUE OF WHETHER OR NOT ALL NODES BETWEEN
        CURRENTNODE AND LASTOCCURRENCE HAVE ONLY ONE SUCCESSOR;
   BEGIN
   LOGICAL TRUTH;
   INTEGER POINTER;
   TRUTH:=TRUE;
   POINTER:=PREDECESSOR(CURRENTNODE);
   WHILE  ((POINTER ¬= LASTOCCURRENCE) AND (TRUTH = TRUE)) DO
       BEGIN
       IF NUMBER_OF_SUCCESSORS(POINTER) > 1 THEN TRUTH:=FALSE;
       POINTER:=PREDECESSOR(POINTER);
       END;
       TRUTH
   END ONE_WAY;

LOGICAL PROCEDURE MATCHING;
COMMENT RETURN LOGICAL VALUE OF WHETHER OR NOT A STRING OF LENGTH
        GIVEN BY DISTANCE(GLOBAL) IS REPEATED TWICE PREVIOUS OF AND
```

```
        INCLUDING CURRENTNODE;
    BEGIN
    LOGICAL MATCH;
    INTEGER P1, P2, COUNTER;
    MATCH:=TRUE;
    COUNTER:=DISTANCE;
    P1:=LASTOCCURRENCE;
    P2:=CURRENTNODE;
    IF COUNTER = 0 THEN MATCH:=FALSE;
    WHILE ((MATCH = TRUE) AND (COUNTER >= 1)) DO
        BEGIN
        IF NAME_OF(P1) ¬= NAME_OF(P2) THEN MATCH:=FALSE
        ELSE
            BEGIN
            COUNTER:=COUNTER - 1;
            P1:=PREDECESSOR(P1);
            P2:=PREDECESSOR(P2);
            IF ((P1 =NIL) AND (COUNTER>0)) THEN MATCH:=FALSE;
            END;
        END;
    MATCH
    END MATCHING;

INTEGER PROCEDURE CHECKBACK;
COMMENT CHECK FOR POSSIBLE LOOPS ON CURRENT PATH. RETURN VALUE INDICATES
        INVALID DIRECTION FROM CURRENTNODE;
    BEGIN
    INTEGER DO_NOT_GO;
    LOGICAL DONE, TOP;
    DO_NOT_GO:=-1;        COMMENT DEFAULT;
    DONE:=FALSE;
    TOP:=FALSE; COMMENT NOT REACHED THE BEGINNING OF THE PATH YET;
    LASTOCCURRENCE:=CURRENTNODE;
    DISTANCE:=0;
    WHILE DONE = FALSE DO
        BEGIN
        IF OCCURS_TWICE(LASTOCCURRENCE,NAME_OF(CURRENTNODE),TOP) THEN
            BEGIN
            IF ((DISTANCE = 1) OR (ONE_WAY)) THEN
                BEGIN
                DO_NOT_GO:=NAME_OF(  SUCCESSOR(LASTOCCURRENCE));
                DONE:=TRUE;
                END
            ELSE
                BEGIN
                IF MATCHING THEN
                    BEGIN
                    DO_NOT_GO:=NAME_OF(  SUCCESSOR(LASTOCCURRENCE));
```

```
                        DONE:=TRUE;
                        END
                    ELSE
                        BEGIN
                        COMMENT IF TOP = FALSE TRY NEXT;
                        IF TOP = TRUE THEN DONE:=TRUE;
                        END;
                    END;
                END
            ELSE
                BEGIN
                DONE:=TRUE; COMMENT CURRENT NODE DOES NOT OCCUR TWICE;
                END;
            END;
        DO_NOT_GO
        END CHECKBACK;

COMMENT             PATH ANALYSIS;

PROCEDURE SET_PATHS;
COMMENT ALGORITHM TO STORE ALL POSSIBLE PATHES HAVING NO REPETATIVE
        SEQUENCE OF OF NODES INTO A SYSTEM OF LINKED PATHS;
    BEGIN
    LOGICAL STOP, FIRSTCHOICE;
    INTEGER ID, WRONG_WAY, I, FATHER;
    INTEGER POINTER;
    INTEGER TEMP;
    ID:=1;
    WHILE CURRENT_PATH_ID  <= NUMBER_OF_PATHS DO
        BEGIN
        STOP:=FALSE;
        CURRENTNODE:=END_OF_PATH(CURRENT_PATH_ID);
        IF NUMBER_OF_SUCCESSORS(CURRENTNODE) = 0 THEN
            BEGIN
            IF TOGGLE = 0 THEN
            LIST_PATH(CURRENTPATH,0);
            STOP:=TRUE;
            LINKFORWARD(CURRENTNODE,NIL);
            CURRENT_PATH_ID:=CURRENT_PATH_ID + 1;
            CURRENTPATH:=NEXT_PATH(CURRENTPATH);
            COMMENT TOGGLE = 0 MEANS:  WRITE EACH PATH IMMEDIATELY.
                    REMOVE STRUCTURE AND SAVE SPACE;
            IF ((TOGGLE = 0) AND (CURRENTPATH ¬= NIL)) THEN
                BEGIN
                POINTER:=PREVIOUS_PATH(CURRENTPATH);
                SET_PATH_ID(POINTER,CURRENT_PATH_ID);
                REMOVE_PATH(POINTER);
                NUMBER_OF_PATHS:=NUMBER_OF_PATHS+1;
```

```
            END;
        END;  COMMENT END OF CURRENT PATH;
    IF ((NUMBER_OF_SUCCESSORS(CURRENTNODE) = 1) AND
        (SON(1,CURRENTNODE) = CHECKBACK)) THEN
        BEGIN
        UPDATE_REACHABILITY(CURRENTNODE,-1);
        STOP:=TRUE;
        REMOVE_PATH(CURRENTPATH);
        CURRENT_PATH_ID:=PATH_ID(CURRENTPATH);
        END
    ELSE
        BEGIN
        LASTOCCURRENCE:=CURRENTNODE;
        DISTANCE:=0;
        BRANCH_POINT:=PREDECESSOR(CURRENTNODE);
        IF BRANCH_POINT = NIL THEN
        BRANCH_POINT:=CURRENTNODE;
        END;
    WHILE STOP = FALSE DO
        BEGIN
        WRONG_WAY:=CHECKBACK;
        I:=1;
        FIRSTCHOICE:=TRUE;
        FATHER:=CURRENTNODE;
        WHILE I <= NUMBER_OF_SUCCESSORS(FATHER) DO
            BEGIN
            IF FIRSTCHOICE = TRUE THEN
                BEGIN
                IF SON(I,CURRENTNODE) ¬= WRONG_WAY THEN
                    BEGIN
                    ADDNODE(CURRENTNODE,FIND_ORIGINAL(SON(I,
                                                  CURRENTNODE)));
                    UPDATE_REACHABILITY(CURRENTNODE,1);
                    LASTOCCURRENCE:=CURRENTNODE;
                    DISTANCE:=0;
                    FIRSTCHOICE:=FALSE;
                    END;  COMMENT END OF FIRSTCHOICE;
                END
            ELSE
                BEGIN
                IF SON(I,FATHER)       ¬= WRONG_WAY THEN
                    BEGIN
                    ALTERNATIVE_WAY(SON(I,FATHER));
                    SAVE_MEMORY(FATHER);
                    END;
                END;  COMMENT HANDLING OF A SUBSEQUENT CHOICE COMPL.;
            I:=I+1;
            END;  COMMENT ALL SONS OF FATHER CONSIDERED;
```

```
IF FIRSTCHOICE = TRUE THEN
    BEGIN
    IF NUMBER_OF_SUCCESSORS(FATHER) ¬= 0 THEN
        BEGIN
        STOP:=TRUE;
        UPDATE_REACHABILITY(CURRENTNODE,-1);
        REMOVE_PATH(CURRENTPATH);
        CURRENT_PATH_ID:=PATH_ID(CURRENTPATH);
        END
    ELSE
        BEGIN
        STOP:=TRUE;
        IF TOGGLE = 0 THEN
        LIST_PATH(CURRENTPATH,0);
        LINKFORWARD(FATHER,NIL);  COMMENT INDICATE LEAFNODE;
        CURRENT_PATH_ID:=CURRENT_PATH_ID + 1;
        CURRENTPATH:=NEXT_PATH(CURRENTPATH);
        COMMENT TOGGLE=0 MEANS: WRITE EACH PATH IMMEDIATELY.
                    REMOVE STRUCTURE AND SAVE SPACE;
        IF ((TOGGLE = 0) AND (CURRENTPATH ¬= NIL)) THEN
            BEGIN
            POINTER:=PREVIOUS_PATH(CURRENTPATH);
            SET_PATH_ID(POINTER,CURRENT_PATH_ID);
            REMOVE_PATH(POINTER);
            NUMBER_OF_PATHS:=NUMBER_OF_PATHS+1;
            END;
        END;
    END;
IF ((NUMBER_OF_SUCCESSORS(CURRENTNODE) = 1) AND
    (SON(1,CURRENTNODE) = CHECKBACK)) THEN
    BEGIN
    STOP:=TRUE;
    UPDATE_REACHABILITY(CURRENTNODE,-1);
    CORRECT_REACHABILITY;
    IF ((TOGGLE = 0) AND (CURRENTPATH = LASTPATH)) THEN
        BEGIN
        COMMENT DONT REMOVE LAST PATH;
        CURRENT_PATH_ID:=CURRENT_PATH_ID +1;
        WRITE(" ")
        END
    ELSE
        BEGIN
        REMOVE_PATH(CURRENTPATH);
        CURRENT_PATH_ID:=PATH_ID(CURRENTPATH);
        END;
    END;
  END;  COMMENT END OF A PATH;
END;  COMMENT ALL PATHS SET;
```

```
    END SET_PATHS;

PROCEDURE SAVE_MEMORY(INTEGER VALUE POINTER);
COMMENT REMOVES LASTPATH WHENEVER IT CANNOT BE CONTINUED;
    BEGIN
    INTEGER SAVEC, SAVEL, SAVED, SAVECP, SAVECPI;
    SAVEC:=CURRENTNODE;
    SAVEL:=LASTOCCURRENCE;
    SAVED:=DISTANCE;
    SAVECP:=CURRENTPATH;
    SAVECPI:=CURRENT_PATH_ID;
    CURRENT_PATH_ID:=PATH_ID(LASTPATH);
    CURRENTPATH:=LASTPATH;
    CURRENTNODE:=END_OF_PATH(PATH_ID(LASTPATH));
    IF ((NUMBER_OF_SUCCESSORS(CURRENTNODE) = 1) AND
        (SCN(1,CURRENTNODE) = CHECKBACK)) THEN
        BEGIN
        UPDATE_REACHABILITY(CURRENTNODE,-1);
        REMOVE_PATH(LASTPATH);
        END
    ELSE
        BEGIN
        BRANCH_POINT:=POINTER;
        END;
    CURRENTNODE:=SAVEC;
    LASTOCCURRENCE:=SAVEL;
    DISTANCE:=SAVED;
    CURRENTPATH:=SAVECP;
    CURRENT_PATH_ID:=SAVECPI;
    END SAVE_MEMORY;


PROCEDURE CORRECT_REACHABILITY;
COMMENT  DUE TO REMOVAL OF THE CURRENT PATH THE REACHABILITY OF ALL
         PREVIOUS NODES UP TO AND EXCLUDING THE MOST RECENT BRANCH POINT
         HAS TO BE CORRECTED;
    BEGIN
    INTEGER POINTER;
     POINTER:=PREDECESSOR(CURRENTNODE);
    WHILE POINTER ¬= BRANCH_POINT DO
        BEGIN
        UPDATE_REACHABILITY(POINTER,-1);
        POINTER:=PREDECESSOR(POINTER);
        IF POINTER = NIL THEN
        WRITE("WARNING: POSSIBLE ERROR IN REACHABILITY CALCULATION.");
        END;
    END CORRECT_REACHABILITY;
```

```
PROCEDURE UPDATE_REACHABILITY(INTEGER VALUE POINTER, N);
COMMENT UPDATE THE REACHABILITY OF THE ORIGINAL NODE POINTED AT BY
        POINTER USING THE VALUE OF N;
   SETCAR1(CDR2(CAR2(POINTER+1)),CAR1(CDR2(CAR2(POINTER+1))) + N);

PROCEDURE LIST_REACHABILITY;
COMMENT LIST ALL NODES OF THE DIRECTED GRAPH AND THEIR CORRESPONDING
        REACHABILITIES;
   BEGIN
   INTEGER I, NEXT;
   REAL SUM, INDEX;
   SUM:= 0.0;
   SPACE(1);
   WRITE("NUMBER OF NODES:",NUMBER_OF_NODES);
   WRITE("NUMBER OF ARCS :",NUMBER_OF_ARCS);
   WRITE("NUMBER OF PATHS:",NUMBER_OF_PATHS);
   WRITE("CYCLOMATIC NUMBER:  V(G)=",NUMBER_OF_ARCS-NUMBER_OF_NODES+2);
   SPACE(2);
   WRITE("REACHABILITY OF NODES:");
   NEXT:=2;
   I:=1;
   WHILE I <= NUMBER_OF_NODES DO
       BEGIN
       WRITE("NODE",NAME_OF(NEXT),":",CAR1(CDR2(NEXT)));
       SUM:=SUM + CAR1(CDR2(NEXT));
       NEXT:=NEXT+K2;
       I:=I+1;
       END;
   WRITE("                   --------");
   WRITE("     SUM:",SUM);
   WRITE("REACHABILITY INDEX");
   WRITE("OF DIRECTED GRAPH:",SUM/NUMBER_OF_NODES);
   END LIST_REACHABILITY;


PROCEDURE LIST_PATH(INTEGER VALUE POINTER, TOGGLE);
COMMENT LIST NODES OF THE PATH POINTED AT ITS PATH HEADER BY POINTER.
        IF TOGGLE IS 0 THEN PRECEEDE THE LIST BY THE ID FOUND IN THE
        PATHHEADER OTHERWISE USE GLOBAL VARIABLE PATH_PRINTED;
   BEGIN
   INTEGER ID, NEXT, COUNTER;
   INTFIELDSIZE := 4;
   PATHS_PRINTED:=PATHS_PRINTED + 1;
   COUNTER:=0;
   IF TOGGLE = 0 THEN ID:=PATH_ID(POINTER) ELSE ID:=PATHS_PRINTED;
   IF POINTER = NIL THEN ERROR(11,0);
   SPACE(1);
```

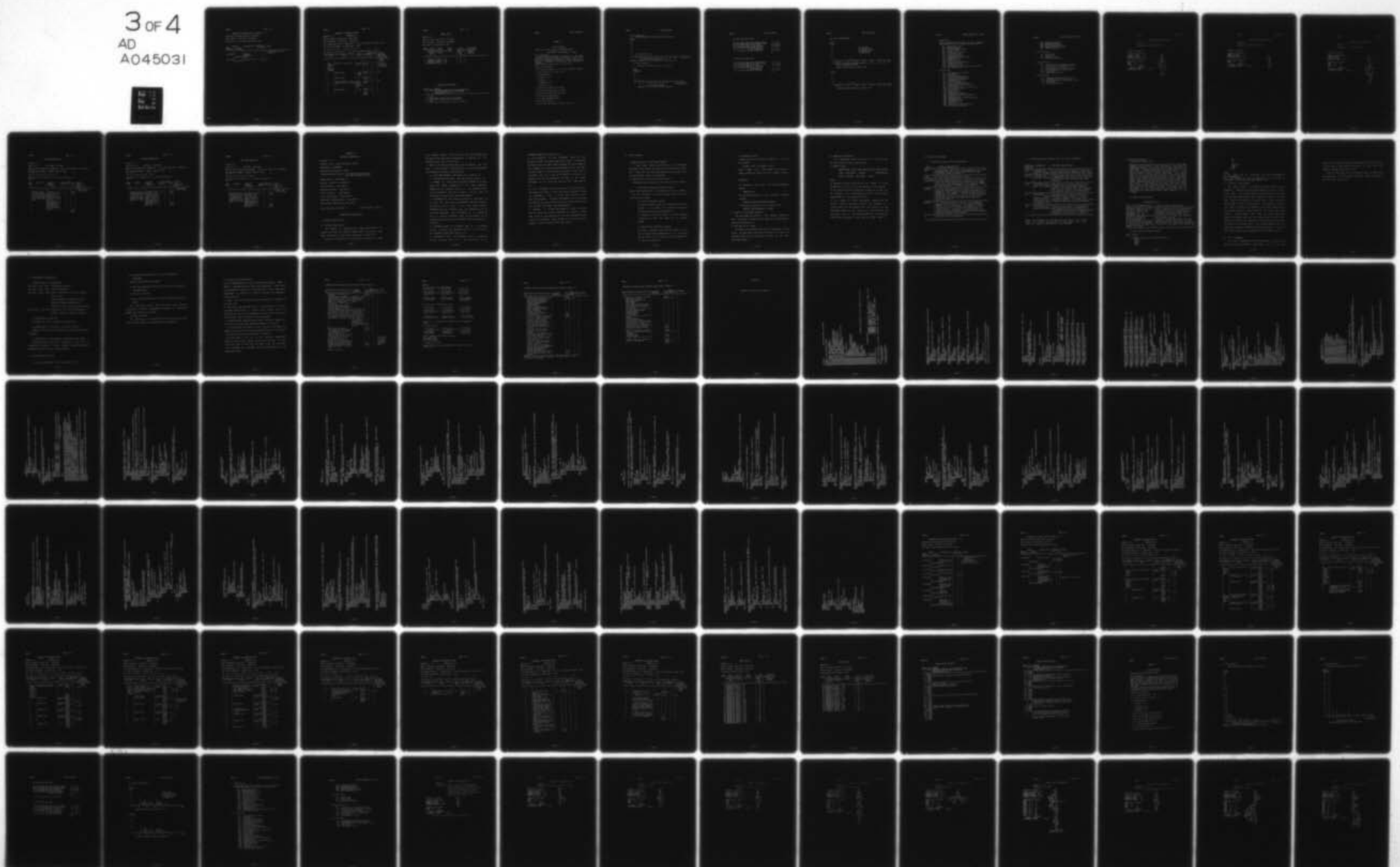
```
    WRITE("PATH",ID,":");
    INTFIELDSIZE := MAXSIZE;
    NEXT:=SUCCESSOR(POINTER);
    IF NEXT = NIL THEN ERROR(12,0);
    WHILE ((NEXT ¬= 1) AND (NEXT ¬= 0)) DO
        BEGIN
        WRITEON(NAME_OF(NEXT));
        NEXT:=SUCCESSOR(NEXT);
        COUNTER:=COUNTER + 1;
        END;
    INTFIELDSIZE := 4;
    WRITEON("...(",COUNTER,"NODES VISITED)");
    IF COUNTER  <  2  THEN
        BEGIN
        WRITE("WARNING :  PATH OF LENGTH 0 . CHECK INPUT.");
        END;
    INTFIELDSIZE := 5;
    END LIST_PATH;


PROCEDURE LIST_ALL_PATHS;
COMMENT LIST ALL PATHS STORED IN THE SYSTEM;
    BEGIN
    INTEGER I, NEXT;
    I:=1;
    NEXT:=PATHHEAD;
    WHILE I <= NUMBER_OF_PATHS DO
        BEGIN
        LIST_PATH(NEXT,0);
        I:=I+1;
        NEXT:=NEXT_PATH(NEXT);
        END;
    END LIST_ALL_PATHS;



PROCEDURE ANALYZE;
COMMENT ANALYZE NEXT DIRECTED GRAPH;
    BEGIN
    SPACE(3);
    WRITE("THE FOLLOWING INPUT DESCRIBES THE DIRECTED GRAPH.");
    SPACE(3);
    INITIALIZE_ALL;
    INPUT_DIRECTED_GRAPH_INFORMATION;
    INITPATHLIST;
    IF TOGGLE = 0 THEN WRITE("RESULTS OF PATH ANALYSIS:");
    IF TOGGLE = 0 THEN WRITE("(PATHS ARE PRINTED DURING ANALYSIS.)");
```

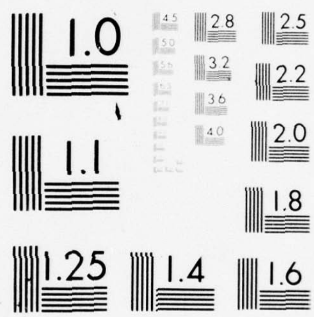
```
    SET_PATHS;
    SPACE(2);
    IF TOGGLE ¬= 0 THEN
        BEGIN
        WRITE("RESULTS OF PATH ANALYSIS:");
        SPACE(2);
        LIST_ALL_PATHS;
        END;
    IF PATHS_PRINTED = 0 THEN WRITE("NO PATH FOUND.");
    SPACE(2);
    WRITE("COMPLEXITY MEASURES:");
    LIST_REACHABILITY;
    SPACE(1);
    WRITE("**************************************************************");
    WRITEON("*******************************************");
    END ANALYZE;

PROCEDURE READ_HEADLINE;
COMMENT READ FIRST CARD OF NEXT DIRECTED GRAPH INFORMATION;
    BEGIN
    STRING(80) CARDBUFFER;     COMMENT TEXTBUFFER (HEADLINE);
    READCARD(CARDBUFFER);
    IF CARDBUFFER(0|1) ¬= "1" THEN MAINSWITCH:= -1
    ELSE
        BEGIN
        MAINSWITCH:=1;
        SPACE(1);
        WRITE(" "); WRITEON(CARDBUFFER(1|79));
        END;
    END READ_HEADLINE;

COMMENT             MAIN;
ON:=1;
WRITE("ANALYSIS OF DIRECTED GRAPHS:");
WRITE("VERSION B. (4/17/77)     WRITTEN BY  HEINZ-MICHAEL HOFFMANN");
SPACE(3);
TERMINATE:
READ_HEADLINE;
WHILE  MAINSWITCH = ON DO
    BEGIN
    ANALYZE;
    READ_HEADLINE;
    END;
WRITE("ALL DIRECTED GRAPHS ANALYZED. END OF EXECUTION. ");
OVERFLOW:
END.
```

AD-A045 031 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
AN EXPERIMENT IN SOFTWARE ERROR OCCURRENCE AND DETECTION.(U)
JUN 77 H HOFFMANN

UNCLASSIFIED NL

3 OF 4
AD
A045031





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

WORKSHEET FOR CODING PHASE OF PROJECT # : 3

Beginning of Coding (day/time) : 3/20/2000

End of Coding (day/time) : 3/21/1200

Man hours : 4.0 (including punching of cards)

CODING BEGIN DAY/TIME	END DAY/TIME	PROGRAM PART	ERROR[1] #	DAY TIME	COMMENT
					1) Record when error is detected.
03/20/2000		-Changes -Subroutines (new)			
	03/20/2200				
03/21/1000		-Punching cards	1	1050	C19
	03/21/1200				

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 3 DEBUG Run # : 1

Begin of Debug Run (day/time) : 03/21/1200

End of Debug Run (day/time) : 03/21/1520

# of Debug Steps incl. in Debug Run: 5 CPU time for Debug run (sec): 41.13

CPU time for necessary compiles (sec) : 29.98

a) 6.03 b) 8.43 c) 7.65 d) 7.87 e) f) g)

Man hours for this Debug Run : 3.0 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	New subroutines and program changes	-Get error free compile	1 compile error	3/21 1215	1.0	2	C6 (forgotten declaration of variable)
2		-repeat step 1	1 program error	3/21 1315	0.5	3	C24
3		-repeat step 1	O.K.	3/21 1345	0.3		
4		-Check reachability calculation for trivial cases	wrong branchpoint calculation	3/21 1400	0.7	4	C21
5		-repeat step 4	O.K.	3/21 1450	0.5		
				3/21 1520			

ERROR LISTING

PROJECT # : 3

Begin of Project (day/time) : 03/19/1500

End of Project (day/time) : 03/23/2300

Man hours for total project : 33.0

ERROR #	PHASE in which ERROR was dis-covered	PHASE in which ERROR was made	ERROR TYPE (see ANNEX F)	TIME spent to solve the ERROR (Man min.)	# of OTHER STATEMENTS OR PARTS OF THE PROGRAM AFFECTED
1	Coding	Coding	C19	10	
2	Debugging	Coding	C6	5	
3	Debugging	Coding	C24	5	
4	Debugging	Coding	C21	20	

ERROR LISTING (COMMENTS)

ERROR #	DAY TIME	COMMENTS (EVIDENCE, THOUGHTS, WHY WAS THE ERROR MADE? WHY AND HOW WAS THE ERROR DISCOVERED? ERROR BLOCKING, etc.)
	03/21	
1	1115	Checking code while punching cards.
2	1215	Found through compiler diagnostics.
3	1315	
4	1400	Found by interpreting results of debug run.

PROJECT # 3

FINAL STATISTICS

Project name : PATH ANALYSIS IN DIRECTED GRAPHS

WITH RESPECT TO REACHABILITY OF NODES

Short description:
This project is an extension of project # 2. The graph analyzing program developed in project # 2 is modified to calculate the reachability of each individual node. At the end of the analysis the reachability index of each node is printed. All features of the program being changed are preserved.
Input: via punch cards Output: via line printer

Quantitative measures:

1. # of source statements : 70 (including necessary changes)

2. Total man hours for project : 33.0

3. Man hours spent in

   a) Design : 7.0

   b) Coding : 4.0

   c) Debugging : 3.0

   d) Testing : 19.0

4. CPU time for compiles: 54.08 sec.

5. CPU time for debug runs: 41.13 sec.

6. CPU time for test runs: 391.48 sec.

7. # of test and debug runs: 4

8. # of test and debug steps: 8

9. # of errors found: 4

10. Total man hours used to correct errors: 0.7

11. Error Detection:

Mean time between error detections: 43.3 man min.

```
 # of
errors
  ↑
  :
5 +
  :
  +
  :
  +
  :
  +
  :
1 +      *******************
  :      *      *      *      *
  ------+------+------+------+------+------+------+------+------+------> man min of
        1 5    3 0    4 5    6 0    7 5    9 0    105    120    135      project time
            TIME BETWEEN ERROR DETECTIONS
(measured from point in time of detection of previous error)
```

12. Error Correction:

Mean time to correct an error: 10.0 man min.

```
# of
Errors
corrected
 ↑
 :
5+
 :
 :
 :     *
1:     *     *           *
-+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+----------->
 1     5    10    15    20    25    30    35    40                 man minutes
                                                                   of project time
                 TIME TO CORRECT ERRORS
      (measured in from point in time of detection)
```

13. When errors were found:

a) # of errors found during design phase:	0 = 0.0 %
b) # of errors found during design review:	0 = 0.0 %
c) # of errors found during coding:	1 = 25.0 %
d) # of errors found during debugging:	3 = 75.0 %
e) # of errors found during writing of test procedures:	0 = 0.0 %
f) # of errors found during testing:	0 = 0.0 %

	4

14. When errors were made:

a) # of errors made during design phase:	0 = 0.0 %
b) # of errors made during design review:	0 = 0.0 %
c) # of errors made during coding:	4 = 100 %
d) # of errors made during debugging:	0 = 0.0 %
e) # of errors made during writing of test procedures:	0 = 0.0 %
f) # of errors made during testing:	0 = 0.0 %

	4

15. TIME HISTORY GRAPHS :

```
# of
Errors
 ^
 :
 :                                      Error Types:
 :                                      A: Clerical error
 :                                      B: Debugging error
 :                                      C: Coding error
 :                                      D: Design error
5+
 :
 :
1:                     C  C C C
 -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-> man
  1 2 3 4 5         10        15        20        25        30    33   hours
 ...DESIGN.....CODING..DEBUGGING..TESTING..............................
    (software development phases)

      NUMBER OF ERRORS FOUND VS PROJECT TIME

# of
Errors

 ^
 :
5+
 :
 :
 :
1+                     C  C C  C
 -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-> man
  1 2 3 4 5         10        15        20        25        30    33   hours

      NUMBER OF ERRORS CORRECTED VS PROJECT TIME
```

1. Design Errors

The follwing types of errors apply to both categories "System Design Errors" and "Program Design Errors":

D1 : Communication Error
D2 : Design Negligence
D3 : Forgotten Cases or Steps
D4 : Timing Problems
D5 : Errors in I/O Concepts
D6 : Data Design Error
D7 : Initialization Error
D8 : Inadequate Checking
D9 : Extreme Conditions Neglected
D10: Sequencing Error
D11: Indexing Error
D12: Loop Control Errors
D13: Misuse of Boolean Expression
D14: Mathematical Error
D15: Representation Error
D16: Misunderstanding of Problem Specifications
D17: Other Design Errors

2. Coding Errors

C1 : Misunderstanding of Design
C2 : Negligence
C3 : I/O Format Error
C4 : Misplaced Data Declaration
C5 : Multiple Data Declarations
C6 : Missing Data Declaration
C7 : Inadequate Data
C8 : Initialization Error
C9 : Error in Parameter Passing
C10: Inadequate or Forgotten Checking
C11: Level Problems
C12: Missing Declarations of Block Limits
C13: Case selection error
C14: GO TO Problems
C15: Comment Error
C16: Forgotten Delimiter
C17: Inconsistency in Naming
C18: Wrong Use of Nested IF Statements
C19: Indexing Error
C20: Inconsistent Use of Variables or Data
C21: Sequencing Error
C22: Flag Usage Problems
C23: Syntax Error
C24: Loop Control Error
C25: Incorrect Exit from Subroutines
C26: Language Usage Problems

C27: Forgotten Statements
C28: Representation Error
C29: Control Sequence Error
C30: Incorrect Subroutine Usage
C31: Other Coding Errors

3. Clerical Errors

A1 : Manual Error
A2 : Mental Error
A3 : Procedural Errors
A4 : Other Clerical Errors

4. Debugging Errors

B1 : Inappropriate Use of Debugging Tools
B2 : Insufficient or Inappropriate Selection of Test Cases or Test Data
B3 : Misinterpretation of Debugging Results
B4 : Misinterpretation of Error Source
B5 : Negligence
B6 : Other Debugging Errors

5. Testing Errors

T1 : Inadequate Test Case(s) or Test Data
T2 : Misinterpretation of Test Results
T3 : Misinterpretation of Program Specification
T4 : Negligence
T5 : Other Testing Errors

DIRECTED GRAPH REPRESENTATION

PROJECT # : 3

Program part : CORRECT REACHABILITY

COMPLEXITY MEASURES:

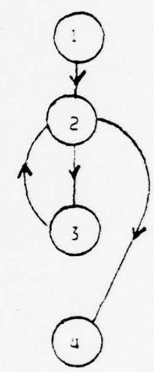
NUMBER OF STATEMENTS: 10

NUMBER OF NODES: 4
NUMBER OF ARCS : 4
NUMBER OF PATHS: 2
CYCLOMATIC NUMBER: V(G)= 2

REACHABILITY OF NODES:

NODE	1	:	1
NODE	2	:	2
NODE	3	:	1
NODE	4	:	2
SUM:			6.000000

REACHABILITY INDEX
OF DIRECTED GRAPH: . 1.500000



DIRECTED GRAPH REPRESENTATION

PROJECT # : 3

Program part : UPDATE REACHABILITY

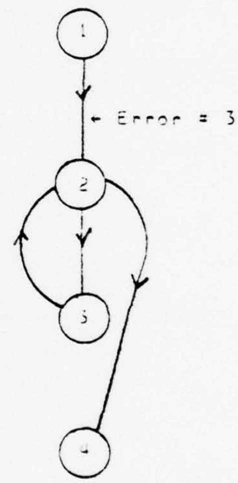
```
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 3
NUMBER OF NODES:      2
NUMBER OF ARCS :      1
NUMBER OF PATHS:      1
CYCLOMATIC NUMBER:  V(G)=    1

REACHABILITY OF NODES:
NODE    1  :    1
NODE    2  :    1
             --------
     SUM:     2.000000
REACHABILITY INDEX
OF DIRECTED GRAPH:        1.000000
```

DIRECTED GRAPH REPRESENTATION

PROJECT # : 3

Program part : LIST REACHABILITY

```
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 13
NUMBER OF NODES:      4
NUMBER OF ARCS :      4
NUMBER OF PATHS:      2
CYCLOMATIC NUMBER:  V(G)=    2

REACHABILITY OF NODES:
NODE     1  :    1
NODE     2  :    2
NODE     3  :    1
NODE     4  :    2
               ---------
   SUM:        6.000000
REACHABILITY INDEX
OF DIRECTED GRAPH:        1.500000
```



TEST PHASE DESCRIPTION

Project # : 3

Test run # : 1   Including 1 Test Step

Begin of Test (day/time) : 03/22/1000   End of Test (day/time) : 03/22/1800

CPU time for necessary compiles (in sec.): 8.33
a) 8.33 b)   c)   d)   e)   f)   g)

CPU time for TEST run (sec) : 80.74

Man Hours for this Test run : 8.0 (including preparation of tests)

TEST STEP	OBJECTVE	EXPECTED RESULT (TOLERANCE)	ACTUAL RESULT	1) ERROR #	DAY TIME	COMMENTS AND CODED ERROR TYPES
						1) Record when error occurs.
1	Compare results of previous test runs with results of modified version of the program using same input data.	Program performance should not be changed with respect to path analysis as being specified in project # 2. Additional output (such as cyclomatic number and reachability index) must be verified by desk checking.	O.K.		3/22 1000 – 3/22 1800	

TEST PHASE DESCRIPTION

Project # : 3

Test run # : 2 Including 1 Test Step

Begin of Test (day/time) : 03/22/1800 End of Test (day/time) : 03/23/1800

CPU time for necessary compiles (in sec.): 7.90
a) 7.90 b) c) d) e) f) g)

CPU time for TEST run (sec) : 139.00

Man Hours for this Test run : 8.0 (including preparation of tests)

TEST STEP	OBJECTVE	EXPECTED RESULT (TOLERANCE)	ACTUAL RESULT	1) ERROR #	DAY TIME	COMMENTS AND CODED ERROR TYPES
						1) Record when error occurs.
1	Compare hand calculated results of NTDS subroutines of module 1 with program results. Check for discrepancies (if any).	In cases of discrepancies program should provide the correct answer otherwise results must match. In case of discrepancies results must be verified by desk checking.	O.K.		3/22 1800 3/23 1800	

TEST PHASE DESCRIPTION

Project # : 3

Test run # : 3   Including 1 Test Step

Begin of Test (day/time) : 03/23/2000   End of Test (day/time) : 03/23/2300

CPU time for necessary compiles (in sec.): 7.87
a) 7.87 b)   c)   d)   e)   f)   g)

CPU time for TEST run (sec) : 171.74

Man Hours for this Test run : 3.0 (including preparation of tests)

TEST STEP	OBJECTVE	EXPECTED RESULT (TOLERANCE)	ACTUAL RESULT	ERROR # [1)]	DAY TIME	COMMENTS AND CODED ERROR TYPES
						1) Record when error occurs.
1	Compare hand calculated results of NTDS subroutines of module 2 with program results. Check for discrepancies (if any).	In cases of discrepancies program should provide the correct answer otherwise results must match. In case of discrepancies results must be verified by desk checking.	O.K.		3/23 2000 3/23 2300	

APPENDIX D

PROJECT DESCRIPTION

Project # : 4

Project title: DATA RETRIEVAL SYSTEM

Programmer : HOFFMANN

Programming Language : ALGOL

Programming environment: IBM/360/67,OS/MVT,BATCH
and TIME SHARING (CP/CMS)

Design notes : see ANNEX A

Program listing : see ANNEX B

Coding notes : see ANNEX C

Debugging notes : see ANNEX D

Error Listing : see ANNEX E

Final statistics : see ANNEX F

Graphical representation : see ANNEX G

Test phase description: see ANNEX H

Starting date: 19 APRIL 77 Ending date: 3 MAY 77

EXPERIMENT DESCRIPTION

1. Project description:

A. General Description

The program is designed for usage under CP/CMS. It expects an input file labeled "DBASE INPUT" to contain data base information of the format described in section B.

During initialization all data base information is read

into program memory. After this the user may operate upon the data using functions as described in section C. All functions are input via terminal.

Most of the terminology being used throughout the project is non standard. Therefore the following explanation is provided for better understanding:

a) DATA BASE MEMBER: see description under B. 1.

b) ATTRIBUTE VALUE PAIR: see description under B. 2.

c) MEMBER ID: The program assigns a MEMBER ID to each data base member depending on the input sequence. MEMBER IDs are integers (1,2,3 ...). The MEMBER ID cannot be changed by the user, however, it may be used to reference any particular data base member.

d) ATTRIBUTE ID: The program assigns an ATTRIBUTE ID to each new and distinct ATTRIBUTE NAME being entered during the initial input of the data base members. ATTRIBUTE IDs are integers (1,2,...64) depending also on the input sequence. Each ATTRIBUTE ID is associated which a unique ATTRIBUTE NAME and provides a way of referencing any particular ATTRIBUTE NAME (see section C "LISTA").

e) ATTRIBUTE NAME: An ATTRIBUTE NAME is a character string of length 1-64 preceding the ":" in the input of an ATTRIBUTE VALUE PAIR (see B. 2.).

f) ATTRIBUTE VALUE: An ATTRIBUTE VALUE is a character string following the ":" in the definition of an

ATTRIBUTE VALE PAIR (see B. 2.).

g) KEY ATTRIBUTE: The KEY ATTRIBUTE refers to the second distinct attribute name being input. (The second attribute name has been chosen because the main usage of the program will have data base members which are identified uniquely by their second attribute value pair.) Therefore the ID of the KEY ATTRIBUTE is initially 2, however, this may be changed by the user to any other ATTRIBUTE ID by using the function "KEY" (see section C).

The KEY ATTRIBUTE is not always the "key attribute" in the sense being used in most data bases, because it does not necessarily allow a unique identification of all data base members. (Instead the MEMBER ID can be viewed as internal key which always allows a unique identification of each individual data base member.)

h) CONTROL: If a data base member is taken into control by using the function "CONTROL" an internal reference is set by the program which allows subsequent examination of ATTRIBUTE VALUE PAIRS of this particular member by using functions "FA" and "ATTR".

B. Input Format:

1. Description of a Data Base Member:

A data base member is defined by 1 - 64 different attribute value pairs followed by the termination symbol "#". After the last data base member an additional termination symbol has to be inserted.

2. Description of Attribute Value Pairs:

Each attribute value pair has the following format:

<attribute name>:<attribute value>#

where "attribute name" is a character string of length 1 - 63 not including ":" and "attribute value" is one of the following types:

a) <single discrete value>

A single discrete value is a character string of length 0 - n, where n is limited by the amount of memory available (see section D).

A character string of length 0 will be replaced internally by "***" and specifies an unknown value.

b) <$<multiple discrete values>>

Multiple discrete values are any number of single discrete values separated by ",". The total string length of all values including separating "," may not exceed 64.

c) <@<range value>>

A range value is a string of length 6 - 64 of the format:

<lower limit> TO <upper limit>

where "lower limit" and "upper limit" are integer values and <lower limit> <= <upper limit>.

Examples:

a) Attribute value pair with single discrete value:

NAME:SMITH#

b) Attribute value pair with multiple dicrete values:

CHECKING ACCOUNT:$804020,50033#

c) Attribute value pair with range value:

MONTHLY PAYMENTS:@550 TO 725#

3. Number of data base members:

The data base may contain 1 -150 members depending upon the amount of memory being used for the storage of values (see section D).

4. Termination of input:

In order to terminate the input of data base information an additional termination symbol ("#") has to be inserted after the the termination symbol of the last data base member.

5. Reserved characters:

All characters other than ":", ",", "#", "$", and "@" may be used without restriction.

6. Sample input:

```
NAME:MEYER# FIRST NAME:JOE# AGE:57## NAME:SMITH#
FIRST  NAME:MIKE#   AGE:##          NAME:NEWMAN#
FIRST NAME:MARY  ANN# AGE:   34###
```

NOTE:

a) Between attribute value pairs any number of blank characters (0,1,2,...) will be ignored by the program.

b) Any number of blank characters preceding a value will be ignored. (In the example the age of the last data base member will be stored as "34".)

c) Any number of blank characters greater than one within a value or attribute name will be reduced to a single blank. (In the example the first name of the last data base member will be stored as "MARY ANN"and the attribute name of the second member will be stored as "FIRST NAME" although it was input as "FIRST NAME" .)

C. Function Commands:

1. Functions without input parameters:

INPUT	FUNCTION DESCRIPTION
LISTA	All attribute names are listed together with their ATTRIBUTE ID. (For usage of ATTRIBUTE ID see functions "ATTR" and "KEY".)
LISTC	All available commands are listed.
LISTDBASE	All data base members are listed printing all of their attribute value pairs together with MEMBER ID of each data base member. (For usage of MEMBER ID see functions "CONTROL" and "LISTM".)
LISTALL	All data base members are listed printing only their KEY ATTRIBUTE and its value preceded by the MEMBER ID. (For usage of MEMBER ID see functions "CONTROL" and "LISTM" . The KEY ATTRIBUTE is the second attribute of the first data base member by default and may be changed by using the function "KEY".)
SWITCH	This allows the user to direct the subsequent output from terminal to a file labeled "DBASE OUTPUT" or or vice versa. To change the output from terminal to file might be appropriate if large output is expected. The contents of the output file may be obtained using the CMS command "OFFLINE PRINTCC DBASE OUTPUT".
&	Termination of the program.

2. Functions which require only one input parameter:

INPUT OF FUNCTION	PARAMETER	FUNCTION DESCRIPTION
LISTM	<MEMBER ID> must be integer	All attribute value pairs of data base member specified by MEMBER ID are listed.
CONTROL	<MEMBER ID> must be integer	Program takes data base member specified by MEMBER ID into control. (For usage of this function see functions "FA" and "ATTR".)
ATTR	<ATTRIBUTE ID> must be integer	List value of attribute specified by ATTRIBUTE ID of data base member in control. (To select proper ATTRIBUTE ID see function "LISTA". To take a member into control see function "CONTROL".)
FA	<ATTRIBUTE NAME> must be a character string of length 1-63 followed by ":"	List value of attribute specified by ATTRIBUTE NAME of member in control. (To take a member into contol see function "CONTROL". To select an existing ATTRIBUTE NAME see function "LISTA".)
KEY	<ATTRIBUTE ID> must be integer	Change key attribute of data base to attibute specified by ATTRIBUTE ID. (For usage of key attribute see functions "LISTALL" and "FIND ID". To select an ATTRIBUTE ID see function "LISTA".)

Note: If parameter is required to be an integer any other input will force the program to terminate. Other wrong input will cause an appropriate error message.

3. FIND and FIND ID:

a) Function Descrition:

These two functions allow the user to find all data base members which satisfy 1 thru 4 conditions specified by subsequent input. (By a condition is meant an attribute value pair. A member satisfies the condition only if it owns this particular attribute value pair or if it owns the attribute without a value specified. The latter case will be indicated by printing "***#" as attribute value in the output following a query.) After the conditions have been input the program tests all data base members as to whether they satisfy all conditions. Only those members which satisfy all conditions of the query are listed. Using "FIND" all of their attribute value pairs are printed whereas using "FIND ID" only the key attribute and its value is printed. (To change the key attribute of the data base see function "KEY".) If no conditions are entered all data base members will be listed.

b) Input of Conditions:

CONDITIONS 1-4:

Input of <ATTRIBUTE NAME>	Input of <attribute value>
ATTRIBUTE NAME can be any string of length 1-63 followed by ":" (it is necessary that the ATTRIBUTE NAME matches with one of the existing ATTRIBUTE NAMES) If an ATTRIBUTE NAME cannot be found an error message is printed and the user can repeat the input.	within queries attribute values are restricted to be 1-64 characters. (If the value stored is longer than 64 characters the comparison is only carried out up to the last character of the query value.) Since all values are stored without preceding blank characters the query value should not have blank characters in front of the value to be tested for.

c) Termination of Query Input:

User inputs:
Q

d) Sample Input for FIND and FIND ID:

```
FIND
NAME:
SMITH
AGE:
```

```
22
Q

FIND ID
AGE:
104
Q
```

NOTE:
1. All input for all functions must not be preceded by blank characters.
2. Each input has to be on a separate line. The input is entered by hitting the carriage return key.

D. Limits of the Program:

The total number of different attribute names may not exceed 64. Assuming 100 data base members having 64 attributes each, the overhead used for implementation (65 items for each data base member) would reduce the amount allocatable items of M2 (20,000) to 13,500 items. Each item of M2 can be used to store 1-4 characters. The number of remaining items divided by the number of data base members gives the average number of items available for each data base member (approximately 135). Thus the total amount of remaining storage locations allow approximately 54,000 characters for values which means that the average string length of values will be less than 8. The program assumes that at least two different ATTRIBUTE NAMEs are entered. If this is not true the user has to change the KEY ATTRIBUTE to "1".

E. Error Messages:

All error messages are self explanatory. Error #1 and #2 will cause the program to terminate. This will be the

case when all memory locations are exhausted (error #1) during the input of data base information or more than 1000 members are specified (error # 2).

Error #3 indicates that an attribute name is longer than 63 characters. (In this case the problem might be solved by examining the input file.)

2. Programmer's background:

a) Experience in programming:

Oct 1970 - May 1971	Programming courses
May 1971 - April 1972	Module Programmer
May 1972 - June 1974	Work in Test and Simulation Department at the NAVAL COMMAND AND CONTROL SYSTEMS COMMAND (FEDERAL GERMAN NAVY) Testing of tactical real time systems
March 1975 - Jan 1977	Student at the NAVAL POSTGRADUATE SCHOOL, Monterey, Computer Science

b) Experience in testing:

Two years of work in testing and simulation.

c) Experience in the area of the given problem:

Usage of similar data structures in previous programming projects.

d) Experience in the progamming language being used:

Experience over a period of 18 months in more than 10 programming projects. (Total number of source statements produced during that time was more 5000.)

3. Psychological factors:

a) Did the programmer like the project? Yes.

b) How does the programmer like the programming language?

Favorite programming language.

c) Was the programmer satisfied by the way the problem was specified?

Only minor criticism.

d) How did the programmer like the programming environment?

The facilities (study room, card punch room) were not conducive to efficient programming because of restricted space, bad lighting and noise.

e) Other factors:

The recording of the experiment's data during the project affected speed and concentration considerably.

4. Comments on Documentation

For the documentation of each software development phase a special documentation form has been developed. These forms are designed to provide a firm guideline for the experiment programmer to record all data of interest for subsequent error analysis.

- Begin and end of each step was recorded with respect to day and time.
- Each error was recorded when it is discovered. The error was then identified by a unique error number (1,2,...). Furthermore the time of discovery and the error type (using error types listed in ANNEX F) were recorded.
- If appropriate, comments about error discovery, reason why the error was made, etc. were documented in ANNEX E2.
- For each error the phase in which the error was made, the phase in which the error was discovered and the time spent to correct the error was recorded in ANNEX E1.
- For each step in any one of the software development phases the day/time of begin and end was recorded. In addition, the time (in man hours) for each step was recorded. This excludes the overhead used for documentation of the experiment data.

WORKSHEET FOR DESIGN PHASE AND DESIGN REVIEW PHASE OF PROJECT # 4

STEP #	PROBLEM AND PLANNED SOLUTION	ALTERNATE SOLUTIONS	DAY TIME	MAN HOURS /STEP	ERROR #	COMMENT
1	Analysis of requirements of project. -Since almost all expected values are of different length usage of linked lists for storage of all attribute values seems to be appropriate. -Study of I/O capabilities under CP/CMS. -View of data base members: Each data base member has a unique identifier (member #). For each member max. 64 different attribute value pairs can be defined by input. Maximum string length for any attribute is 64. The total number of different attributes in the data base may also not exceed 64 (arbitrarily chosen limit).	Storage in fixed lengths string arrays would waste too much memory. a) Let number of different attributes be an input parameter. b) Use a hash table to store attributes. This would make string lengths of attributes independent of the array size of the attribute table.	4/19 1400 4/19 1900	5.0		
2	Define data structures: -Attributes are identified by a character string of length 64 and stored in a string array (ATTRIBUTE). Each attribute can be uniquely identified by its position within the attribute array (ATTRIBUTE ID). -Values of attributes are stored in a linked list structure using a BITS ARRAY (M2). (Length of M2 (124,000 bytes) limits the amount of information to be stored in the data base.)		4/19 1900 4/19 0900	4.0		Primitives for data structures could be copiedfrom project #2.

Remarks: see page 1a

Remarks:

a) Linked list of data base members:

```
|-----|-----|           |-----|-----|           |-----|-----|
|     |    1|           |     |    2|           |     |    n|
|-----|-----|           |-----|-----|           |-----|-----|
| ptr |ptr------------> | ptr |ptr------> ...   | ptr | NIL |
|--|--|-----|           |--|--|-----|           |--|--|-----|
   |                       |                       |
   V                       V                       V
attribute value         attribute value         attribute value
list of member 1        list of member 2        list of member n
                                                (last member)
```

b) Structure of attribute value lists:

```
|-----|-----|            |-----|-----|            |-----|-----|
|     |   A1|            |     |   A2|            |     | A(n)|
|-----|-----|            |-----|-----|            |-----|-----|
| ptr |ptr------------>  | ptr |ptr-------> ...   | ptr | NIL |
|--|--|-----|            |--|--|-----|            |--|--|-----|
   |                        |                        |
   V                        V                        V
value structure          value structure          value structure
of first attribute       of second attribute      of nth attribute
```

where A1, A2,... represent the ATTRIBUTE IDs of corresponding values

c) Value structure:

```
|-----|-----|            |-----|-----|            |-----|-----|
|    t |ptr------------> |     t|ptr--------->... |    t| NIL |
|-----------|            |-----------|            |-----------|
|VALUE REPRE|            |SENTATION..|            |...#.......|
|-----------|            |-----------|            |-----------|
```

where t represents the type of the value:
t=0: single value
t=1: range values
t=2: multiple values

is used as delimiter.

Usage of "****#" for value representation indicates that value is not known.

WORKSHEET FOR DESIGN PHASE AND DESIGN REVIEW PHASE OF PROJECT # 4

STEP #	PROBLEM AND PLANNED SOLUTION	ALTERNATE SOLUTIONS	DAY TIME	MAN HOURS /STEP	ERROR #	COMMENT
3	a) Define utilities for input		4/20			
	of data base information:		0900	4.0		
	-FILL,EXTRACT,GETCHAR,SKIP					
	-GET OLD FILE INFORMATION					
	-STORE,CLEARIN,CLEAROUT					
	-CARDIN,PRINTOUT					
	-TYPIN,TYPOUT					
	-FILE I O,TERMINAL I O					
	-GET ATTRIBUTES AND VALUES					
	-GET ATTRIBUTE,GET VALUE					
	-FIND ATTRIBUTE					
	b) other utilities:					
	-SPACE,ERROR,DIAGNOSTICS		4/20			
	-INITIALIZE ALL		1300			
4	a) Define command capabilities		4/21			
	-List all attributes (LISTA)		0900	2.0		
	-List all members (using only					
	one attribute value pair					
	chosen by user) (LISTALL)					
	-List all members (printing					
	all attribute value pairs of					
	each member (LISTDBASE)					
	-List all available commands					
	(LISTC)					
	-List all attribute value					
	pairs of a single member					
	(LISTM)					
	-Select a single member for					
	retrieval of particular values					
	of interest (CONTROL)					
	-List a particular attribute					
	value pair of member in con-					
	trol (FA,ATTR)					
	-Find all data base members					
	which with given conditions					
	(FIND, FIND ID)					
	-Switch output from terminal					
	to file and vice versa(SWITCH)					
	-Change key attribute (KEY)					
	-Obtain diagnostics (D)					
	-Super user mode (SU)					
	b) All subroutines which han-		4/21			
	dle commands are named accor-		1100			
	dingly.					

Remarks: The last two functions are not required, however, they can be effectively used for testing purposes. They also support further extensions of the program.

WORKSHEET FOR DESIGN PHASE AND DESIGN REVIEW PHASE OF PROJECT # 4

STEP #	PROBLEM AND PLANNED SOLUTION	ALTERNATE SOLUTIONS	DAY TIME	MAN HOURS /STEP	ERROR #	COMMENT
5	Analysis of implementing FIND (FIND ID) function. Definiton of subroutines: -GET CONDITIONS -GET REQUEST -GET QVALUE (functions which allow user to define queries) -TEST CONDITIONS (evaluation of queries) Define functions to test for single discrete values: -FINDATTRIBUTE -EQUAL,MATCHING -FIND MEMBER Define functions to print results: -WRITE RESULTS -LISTATTRIBUTE(value) All members which satisfy user defined conditions are first stored within a linked list. The result of all comparisions are printed after all data members have been examined.		4/22 1230 4/22 1430	1.5		
6	Design of retrieval functions for multiple values and range values: -COMPARE MANY -LLIMIT,ULIMIT (return upper and lower limit)		4/23 1530 4/25 1100	6.0		
7	Design review and implementation of improvements.		4/27 1300 4/27 2030	1.5		

Annex B

Program Listing of Project # 4

```
COMMENT              DATA RETRIEVAL SYSTEM;

COMMENT DATA DEFINITIONS;
BEGIN
BITS ARRAY M1(0::1001);
BITS ARRAY M2(0::40002);
STRING(80) COMMAND;
STRING(64) ARRAY ATTRIBUTE(1::64);
INTEGER ARRAY ATTR_LENGTH(1::64);
STRING(64) ARRAY REQUEST(1::5);
STRING(64) ARRAY CVALUE(1::5);
INTEGER ARRAY VALUE_LENGTH(1::5);
STRING(80) CARDBUFFER;
STRING(80) ARRAY COMMAND_STACK(1::16);
INTEGER NUMBER_OF_COMMANDS;
STRING(1) LASTCHAR;
INTEGER BP;
INTEGER LIST_POINTER;
INTEGER LAST;
INTEGER MEMBER;
INTEGER NUMBER_OF_ATTRIBUTES;
INTEGER NUMBER_OF_CONDITIONS;
INTEGER FIRST_MEMBER;
INTEGER CURRENT_MEMBER;
INTEGER CURRENT_NODE;
INTEGER CPOINTER;
INTEGER CURRENT_ID;
INTEGER KEY;
INTEGER MAX1, MAX2;              COMMENT POINTERS TO LAST ITEMS OF M1, M2;
INTEGER MAINSWITCH;
INTEGER TOGGLE;
INTEGER FOUT;
INTEGER I;
COMMENT              PARAMETERS;
INTEGER NIL;                     COMMENT INDICATES THE END OF A LINKED LIST;
INTEGER N1;                      COMMENT NUMBER OF ITEMS OF M1;
INTEGER N2;                      COMMENT NUMBER OF ITEMS OF M2;
INTEGER K1;                      COMMENT SIZE OF ITEMS OF M1;
INTEGER K2;                      COMMENT SIZE OF ITEMS OF M2;
INTEGER MAXSIZE;                 COMMENT MAXIMUM SIZE OF NODE NAME;

COMMENT PROCEDURES TO SELECT VARIOUS I/O MEDIA;

PROCEDURE CLEARIN;
    IOCONTROL(1);

PROCEDURE CLEAROUT;
    IOCONTROL(2);
```

```
PROCEDURE CARDIN;
COMMENT READ INPUT FROM FILE LABELLED "DBASE OUTPUT";
    BEGIN
    CLEARIN;
    WRITE(" INPUT FROM FILE"); WRITE(" ");
    IOCONTROL(4);
    END CARDIN;

PROCEDURE TYPIN;
COMMENT INPUT FROM TERMINAL;
    BEGIN
    CLEARIN;
    WRITE(" INPUT FROM TERMINAL"); WRITE(" ");
    IOCONTROL(5);
    END TYPIN;

PROCEDURE PRINTOUT;
COMMENT WRITE TO FILE LABELLED "DBASE  OUTPUT";
    BEGIN
    CLEAROUT;
    IOCONTROL(6);
    END PRINTOUT;

PROCEDURE TYPOUT;
COMMENT WRITE OUTPUT ON TERMINAL;
    BEGIN
    CLEAROUT;
    IOCONTROL(7);
    END TYPOUT;

PROCEDURE FILE_I_O;
COMMENT DIRECT INPUT AND OUTPUT FROM AND TO FILES;
    BEGIN
    CARDIN;
    PRINTOUT;
    END FILE_I_O;

PROCEDURE TERMINAL_I_O;
COMMENT TERMINAL INPUT AND OUTPUT;
    BEGIN
    TYPIN;
    TYPOUT;
    END TERMINAL_I_O;

COMMENT END OF I/O SELECTION PROCEDURES;

PROCEDURE  GET_OLD_FILE_INFORMATION;
```

```
    COMMENT READ INFORMATION FROM OLD  INPUT FILE;
    BEGIN
    CARDIN;
    STORE;
    END GET_OLD_FILE_INFORMATION;
PROCEDURE WRITE_NEW_INFORMATION;
COMMENT WRITE UPDATED DATA TO OUTPUT FILE;
    BEGIN
    PRINTOUT;
    END WRITE_NEW_INFORMATION;


COMMENT             PRIMITIVES;

PROCEDURE INITIALIZE1;
    BEGIN
    INTEGER I;
    MAX1:=(N1-1)*K1;
    FOR I:= 0 STEP K1 UNTIL MAX1 DO SETCDR1(I,I+K1);
    SETCDR1(MAX1,NIL);
    END INITIALIZE1;

PROCEDURE INITIALIZE2;
    BEGIN
    INTEGER I;
    MAX2:=(N2-1)*K2;
    FOR I:= 0 STEP K2 UNTIL MAX2 DO SETCDR2(I,I+K2);
    SETCDR2(MAX2,NIL);
    M2(1):=#00000000;    COMMENT SAFETY MEASURE;
    SETCAR2(0,0);        COMMENT SAFETY MEASURE;
    END INITIALIZE2;

INTEGER PROCEDURE CAR1(INTEGER VALUE X);
COMMENT RETURNS VALUE OF CARFIELD OF ELEMENT X IN M1;
    NUMBER(M1(X) SHR 16);

INTEGER PROCEDURE CAR2(INTEGER VALUE X);
COMMENT RETURNS VALUE OF CARFIELD OF ELEMENT X IN M2;
    NUMBER(M2(X) SHR 16);

INTEGER PROCEDURE CDR1(INTEGER VALUE X);
COMMENT RETURNS VALUE OF CDRFIELD IN M1;
    NUMBER(M1(X) AND #FFFF);

INTEGER PROCEDURE CDR2(INTEGER VALUE X);
COMMENT RETURNS VALUE OF CDRFIELD IN M2;
    NUMBER(M2(X) AND #FFFF);
```

```
PROCEDURE SETCAR1(INTEGER VALUE X,Y);
COMMENT SET CARFIELD OF ELEMENT X IN M1 TO Y;
   M1(X):=(M1(X) AND #FFFF) OR (BITSTRING(Y) SHL 16);

PROCEDURE SETCAR2(INTEGER VALUE X,Y);
COMMENT SET CARFIELD OF ELEMENT X IN M2 TO Y;
   M2(X):=(M2(X) AND #FFFF) OR (BITSTRING(Y) SHL 16);

PROCEDURE SETCDR1(INTEGER VALUE X,Y);
COMMENT SET CDRFIELD OF ELEMENT X IN M1 TO Y;
   M1(X):=(M1(X) AND #FFFF0000) OR BITSTRING(Y);

PROCEDURE SETCDR2(INTEGER VALUE X,Y);
COMMENT SET CDRFIELD OF ELEMENT X IN M2 TO Y;
   M2(X):=(M2(X) AND #FFFF0000) OR BITSTRING(Y);

INTEGER PROCEDURE ALLOCATE1;
COMMENT ALLOCATE NEXT ITEM OF M1;
   BEGIN
   INTEGER A;
   A:=CDR1(0);      COMMENT FREELIST BEGINS AT ZERO;
   IF A = NIL THEN ERROR(1,0);
   SETCDR1(0,CDR1(A));
   M1(A):=#C0C00000;            COMMENT CLEAR ITEM;
   A
   END ALLOCATE1;

INTEGER PROCEDURE ALLOCATE2;
COMMENT ALLOCATE NEXT ITEM OF M2;
   BEGIN
   INTEGER A;
   A:=CDR2(0);      COMMENT FREELIST BEGINS AT ZERO;
   IF A = NIL THEN ERROR(2,0);
   SETCDR2(0,CDR2(A));
   M2(A):=#CCC00000;            COMMENT CLEAR ITEM;
   M2(A+1):=#C000C000;
   A
   END ALLOCATE2;

PROCEDURE FREE1(INTEGER VALUE X);
COMMENT RETURN ITEM TO FREELIST M1;
   BEGIN
   SETCDR1(X,CDR1(0));
   SETCDR1(0,X);
   END FREE1;

PROCEDURE FREE2(INTEGER VALUE X);
```

```
COMMENT RETURN ITEM TO FREELIST M2;
    BEGIN
    SETCDR2(X,CDR2(0));
    SETCDR2(0,X);
    END FREE2;

COMMENT              UTILITIES;

PROCEDURE SPACE(INTEGER VALUE N);
COMMENT WRITE N BLANK LINES;
    BEGIN
    INTEGER I;
    IF FOUT = 0 THEN
    FOR I:=1 STEP 1 UNTIL N DO WRITE(" ");
    END SPACE;

PROCEDURE INITIALIZE_ALL;
COMMENT INITIAL SET UP;
    BEGIN
    COMMENT SET ALL PARAMETERS;
    NIL := 0;
    N1 := 1000;
    N2 := 20001;
    K1 := 1;
    K2 := 2;
    INTFIELDSIZE := 5;
    TOGGLE:=0;
    FOUT:=0;
    COMMENT INITIALIZE LINKED LISTS;
    INITIALIZE1;
    INITIALIZE2;
    BF:=EC;
    MEMBER:=0;
    NUMBER_OF_ATTRIBUTES:=0;
    FIRST_MEMBER:=NIL;
    CURRENT_MEMBER:=NIL;
    CURRENT_NODE:=NIL;
    LIST_POINTER:=NIL;
    LAST:=NIL;
    CPOINTER:=NIL;
    COMMAND:=" ";
    NUMBER_OF_CONDITIONS:=0;
    FOR I:=1 UNTIL 64 DO ATTRIBUTE(I):= " ";
    FOR I:=1 UNTIL 64 DO ATTR_LENGTH(I):=0;
    CARDBUFFER:=" ";
    FOR I:=1 STEP 1 UNTIL 5 DO
        BEGIN
        REQUEST(I):=" ";
```

```
            CVALUE(I):=" ";
            VALUE_LENGTH(I):=0;
            END;
        COMMAND_STACK(1):="LISTA";
        COMMAND_STACK(2):="LISTALL";
        COMMAND_STACK(3):="LISTDBASE";
        COMMAND_STACK(4):="LISTC";
        COMMAND_STACK(5):="LISTM";
        COMMAND_STACK(6):="CONTROL";
        COMMAND_STACK(7):="ATTR";
        COMMAND_STACK(8):="FA";
        COMMAND_STACK(9):="FIND";
        COMMAND_STACK(10):="FIND ID";
        COMMAND_STACK(11):="SWITCH";
        COMMAND_STACK(12):="&";
        COMMAND_STACK(13):="KEY";
        COMMAND_STACK(14):="SU";
        COMMAND_STACK(15):="D";
        COMMAND_STACK(16):=" ";
        NUMBER_OF_COMMANDS:=16;
        KEY:=2;
        TOGGLE:=0;
        MAINSWITCH:=0;    COMMENT TERMINAL I/O;
        TYPOUT;
        WRITE("GENERAL PURPOSE INFORMATION RETRIEVAL SYSTEM.");
        WRITE("WRITTEN BY HEINZ-MICHAEL HOFFMANN. VERSION A(4/28/77)");
        SPACE(2);
        END INITIALIZE_ALL;

    COMMENT              ERROR HANDLING;

    PROCEDURE ERROR(INTEGER VALUE N,INFO);
    COMMENT HANDLES ALL ERRORS AND EXEPTIONAL CASES;
        BEGIN
        INTFIELDSIZE := 5;
        WRITE("ERROR : ",N);
        CASE N OF
            BEGIN
            COMMENT ERROR 1;
                BEGIN
                WRITE("FREELIST 1 EXHAUSTED, EXECUTION ENDS.");
                DIAGNOSE(FALSE,TRUE);
                GO TO OVERFLOW;
                END;
            COMMENT ERROR 2;
                BEGIN
                WRITE("FREELIST 2 EXHAUSTED, ANALYSIS TERMINATED.");
                DIAGNOSE(TRUE,FALSE);
```

```
            END;
        COMMENT ERROR 3;
            BEGIN
            WRITE("ATTRIBUTE OF LENGTH",INFO,"CANNOT BE STORED.");
            GO TO OVERFLOW;
            END;
        COMMENT ERROR 4;
            BEGIN
            WRITE("MEMBER",INFO,"NOT FOUND."); SPACE(1);
            END;
        END;
    END ERROR;

PROCEDURE DIAGNOSE(LOGICAL VALUE TERMINATION,SELECT);
COMMENT DEBUGGING AND TESTING TOOL;
    BEGIN
    INTEGER I;
    SPACE(2);
    WRITE("DIAGNOSTICS:");
    SPACE(1);

    IF SELECT = TRUE THEN
        BEGIN
        WRITE("ELEMENT    CAR1     CDR1");
        FOR I:=CDR1(0) STEP -K1 UNTIL 0 DO  WRITE(I,CAR1(I),CDR1(I));
        WRITE("ELEMENT    CAR2     CDR2");
        FOR I:=CDR2(0)+1 STEP -K1 UNTIL 0 DO WRITE(I,CAR2(I),CDR2(I),
                                                  EXTRACT(I));
        END;

    WRITE("FIRST MEMBER:"); WRITEON(FIRST_MEMBER); SPACE(1);
    WRITE("CURRENT MEMBER:"); WRITEON(CURRENT_MEMBER); SPACE(1);
    WRITE("CURRENT NODE:"); WRITEON(CURRENT_NODE); SPACE(1);
    WRITE("CONTROL POINTER:"); WRITEON(CPOINTER); SPACE(1);
    WRITE("NUMBER OF DATA BASE MEMBERS:"); WRITEON(MEMBER); SPACE(1);
    WRITE("LIST POINTER:");  WRITEON(LIST_POINTER); SPACE(1);
    WRITE("KEY ATTRIBUTE:");WRITEON(KEY); SPACE(1);
    WRITE("MAINSWITCH:",MAINSWITCH); SPACE(1);
    WRITE("TOGGLE:");   WRITEON(TOGGLE); SPACE(1);
    WRITE("FOUT:");  WRITEON(FOUT); SPACE(1);
    WRITE("CARDBUFFER:"); WRITE(" "); WRITE(CARDBUFFER); SPACE(1);
    I:=0;
    WHILE I<BP DO BEGIN WRITEON(" "); I:=I+1; END; WRITEON("*");SPACE(1);
    WRITE("NUMBER OF ATTRIBUTES(TOTAL):");
    WRITEON(NUMBER_OF_ATTRIBUTES); SPACE(1);
    LISTA;
    WRITE("NUMBER OF COMMANDS:");WRITEON(NUMBER_OF_COMMANDS); SPACE(1);
    LISTC;
```

```
    WRITE("COMMAND LINE:"); WRITE(COMMAND); SPACE(1);
    FOR I:=5 STEP -1 UNTIL 1 DO
        BEGIN
        WRITE("REQUEST(",I,"):",REQUEST(I));
        WRITE("QVALUE (",I,"):",QVALUE(I));
        END;
    SPACE(1);
    WRITE("NUMBER OF CONDITIONS:"); WRITEON(NUMBER_OF_CONDITIONS);
    SPACE(1);
    WRITE("LAST MEMBER OF LIST OF RESULTS:");  WRITEON(LAST); SPACE(1);
    WRITE("END OF DIAGNOSTICS");
    WRITE("......................................................");
    SPACE(1);
    IF TERMINATION = TRUE THEN
        BEGIN
        WRITE("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++");
        WRITEON("+++++++++++++++++++++++++++++++++++");
        GO TO TERMINATE;
        END;
    END DIAGNOSE;

PROCEDURE GET_COMMANDS;
COMMENT READ COMMANDS FROM TERMINAL;
    BEGIN
    STRING(80) LINE;
    WHILE TRUE DO
        BEGIN
        IF MAINSWITCH = 1 THEN TYPOUT;
        WRITE(">>");  SPACE(1);
        READCARD(LINE);
        COMMAND:=LINE;
        IF MAINSWITCH = 1 THEN PRINTOUT;
        INTERPRET(FIND_COMMAND);
        END;
    END GET_COMMANDS;


STRING(4)  PROCEDURE EXTRACT(INTEGER VALUE Y);
COMMENT EXTRACT 4 CHARACTER CONTENTS STORED IN CELL Y;
    BEGIN
    STRING(4) SYMBOL;
    INTEGER C, I, J;
    BITS TEMP;
    TEMP:=M2(Y);
    FOR I:=1 STEP 1 UNTIL 4 DO
        BEGIN
        J:=4-I;
        C:=NUMBER(TEMP AND #FF);
```

```
            SYMBOL(J|1):= CODE(C);
            TEMP:=(TEMP SHR 8);
            END;
        SYMBOL
        END EXTRACT;


    PROCEDURE FILL(INTEGER VALUE Y; STRING(4) VALUE SYMBOL);
    COMMENT PUT CHARACTERS INTO CELL Y;
        BEGIN
        STRING(1) NEXT;
        INTEGER I;
        BITS Z;
        FOR I:=0 STEP 1 UNTIL 3 DO
            BEGIN
            M2(Y):=(M2(Y) SHL 8);
            NEXT:=SYMBOL(I|1);
            Z:=BITSTRING(DECODE(NEXT));
            M2(Y):=(M2(Y) OR Z);
            END;
        END FILL;


    STRING(1) PROCEDURE GETCHAR;
    COMMENT GET NEXT CHARACTER FROM INPUT FILE;
        BEGIN
        STRING(1) NEXT;
        BP:=BP+1;
        IF BP>79 THEN
            BEGIN
            READCARD(CARDBUFFER);
            BP:=0;
            END;
        NEXT:=CARDBUFFER(BP|1);
        IF ((LASTCHAR = " ") AND (NEXT = " ")) THEN
            BEGIN
            WHILE NEXT = " " DO
                BEGIN
                BP:=BP+1;
                IF BP = 80 THEN
                    BEGIN
                    READCARD(CARDBUFFER);
                    BP:=0;
                    END;
                NEXT:=CARDBUFFER(BP|1);
                END;
            END;
        LASTCHAR:=NEXT;
```

```
    NEXT
    END GETCHAR;

STRING(1) PROCEDURE SKIP;
COMMENT ADVANCE BP UNTIL IT POINTS TO THE NEXT NON BLANK CHARACTER
        AND RETURN THIS CHARACTER;
    BEGIN
    STRING(1) NON_BLANK;
    NON_BLANK:=" ";
    WHILE NON_BLANK = " " DO NON_BLANK:=GETCHAR;
    NON_BLANK
    END SKIP;

PROCEDURE STORE;
COMMENT GET DESCRIPTION OF ALL DATA BASE MEMBERS AND STORE
        INTO DATA STRUCTURES;
    BEGIN
    INTEGER Y;
    WHILE SKIP ¬= "#" DO
        BEGIN
        MEMBER:=MEMBER+1;
        IF MEMBER = 1 THEN
            BEGIN
            FIRST_MEMBER:=ALLOCATE2;
            CURRENT_MEMBER:=FIRST_MEMBER;
            END
        ELSE
            BEGIN
            Y:=ALLOCATE2;
            SETCDR2(CURRENT_MEMBER,Y);
            CURRENT_MEMBER:=Y;
            END;
        SETCDR2(CURRENT_MEMBER,NIL);
        SETCDR2(CURRENT_MEMBER+1,MEMBER);  COMMENT SET MEMBER ID;
        GET_ATTRIBUTES_AND_VALUES;
        END;
    END STORE;

PROCEDURE GET_ATTRIBUTES_AND_VALUES;
COMMENT GET ALL ATTRIBUTES AND VALUES OF CURRENT MEMBER
        AND STORE INTO DATA STRUCTURES;
    BEGIN
    LOGICAL DONE;
    INTEGER Y;
    CURRENT_NODE:=ALLOCATE2;
```

```
    SETCAR2(CURRENT_MEMBER,CURRENT_NODE);
    DONE:=FALSE;
    WHILE DONE=FALSE DO
        BEGIN
        GET_ATTRIBUTE(DONE);
        IF DONE=FALSE THEN
            BEGIN
            GET_VALUE;
            IF SKIP¬= "#" THEN
                BEGIN
                Y:=ALLOCATE2;
                SETCDR2(CURRENT_NODE,Y);
                SETCDR2(Y,NIL);
                CURRENT_NODE:=Y;
                END;
            END;
        END;
    END GET_ATTRIBUTES_AND_VALUES;


PROCEDURE GET_ATTRIBUTE(LOGICAL VALUE RESULT DONE);
COMMENT GETS THE NEXT ATTRIBUTE OF THE CURRENT MEMBER AND STORES
        IT INTO ATTRIBUTE TABLE;
    BEGIN
    STRING(1) NEXTCHAR;
    STRING(64) ATTR;
    INTEGER I, J, Y;
    ATTR:=" ";
    NEXTCHAR:=CARDBUFFER(BP|1);
    COMMENT CALL OF GET_ATTRIBUTE IS ALWAYS PRECEDED BY SKIP;
    IF NEXTCHAR = "#" THEN DONE := TRUE
    ELSE ATTR(0|1):=NEXTCHAR;
    I:=1;
    IF DONE = FALSE THEN
        BEGIN
        WHILE NEXTCHAR ¬= ":" DO
            BEGIN
            NEXTCHAR:=GETCHAR;
            ATTR(I|1):=NEXTCHAR;
            I:=I+1;
            IF I=64 THEN ERROR(3,I);
            END;
        J:=FIND_ATTRIBUTE(ATTR,I);
        IF J=0 THEN
            BEGIN
            NUMBER_OF_ATTRIBUTES:=NUMBER_OF_ATTRIBUTES + 1;
            J:=NUMBER_OF_ATTRIBUTES;
            ATTRIBUTE(NUMBER_OF_ATTRIBUTES):=ATTR;
```

```
                    ATTR_LENGTH(NUMBER_OF_ATTRIBUTES):=I;
                    END;
                SETCDR2(CURRENT_NODE+1,J);  COMMENT ATTRIBUTE ID;
                SETCDR2(CURRENT_NODE,NIL);
                END;
        END GET_ATTRIBUTE;


    PROCEDURE GET_VALUE;
    COMMENT GET VALUE OF CURRENT ATTRIBUTE AND STORE INTO DATA STRUCTURES;
        BEGIN
        INTEGER I, J, X, Y;
        STRING(4) C;
        STRING(1) CHAR;
        C:="***#";
        X:=ALLOCATE2;
        SETCAR2(CURRENT_NODE,X);
        SETCDR2(X+1,NIL);
        CHAR:=SKIP;
        IF CHAR = "#" THEN FILL(X,C)
        ELSE
            BEGIN
            IF CHAR = "@" THEN SETCAR2(X+1,1);COMMENT RANGE VALUE;
            IF CHAR = "$" THEN SETCAR2(X+1,2);COMMENT MORE THAN ONE VALUE;
            COMMENT SINGLE DISCRETE VALUE IS DEFAULT;
            IF ((CHAR = "@" ) OR (CHAR= "$")) THEN CHAR:=GETCHAR;
            I:=0;
            C:="    ";
            WHILE CHAR ¬= "#" DO
                BEGIN
                C(I|1):=CHAR;
                I:=I+1;
                IF I=4 THEN
                    BEGIN
                    FILL(X,C);
                    C:="    ";
                    Y:=ALLOCATE2;
                    SETCDR2(Y+1,NIL);
                    SETCDR2(X+1,Y);
                    X:=Y;
                    I:=0;
                    END; COMMENT VALUE NEEDS MORE STORAGE SPACE;
                CHAR:=GETCHAR;
                IF CHAR = "#" THEN
                    BEGIN
                    C(I|1):=CHAR;
                    FILL(X,C);
                    END;
```

```
            END;
        END;
    END GET_VALUE;


INTEGER PROCEDURE FIND_ATTRIBUTE(STRING(64) VALUE A;INTEGER VALUE
                                                            LENGTH);
COMMENT  RETURN LOCATION OF SAME ATTRIBUTE WITHIN ATTRIBUTE TABLE.
        IF SAME ATTRIBUTE NOT FOUND THEN RETURN 0;
    BEGIN
    INTEGER J;
    LOGICAL FOUND;
    FOUND:=FALSE;
    J:=0;
    WHILE ((J<NUMBER_OF_ATTRIBUTES) AND (FOUND=FALSE)) DO
        BEGIN
        J:=J+1;
        IF ATTRIBUTE(J) = A THEN FOUND:= TRUE;
        END;
    IF FOUND =FALSE THEN J:=0;
    J
    END FIND_ATTRIBUTE;


INTEGER PROCEDURE FIND_COMMAND;
COMMENT FIND GIVEN COMMAND WITHIN COMMAND STACK. RETURN COMMAND ID;
    BEGIN
    INTEGER ID;
    LOGICAL DONE;
    ID:=0;
    DONE:=FALSE;
    WHILE DONE=FALSE DO
        BEGIN
        ID:=ID+1;
        IF COMMAND_STACK(ID) = COMMAND THEN DONE:=TRUE;
        IF ID = NUMBER_OF_COMMANDS THEN DONE:=TRUE;
        END;
    ID
    END FIND_COMMAND;


PROCEDURE INTERPRET(INTEGER VALUE ID);
COMMENT INTERPRET AND EXECUTE COMMAND OF GIVEN ID;
    BEGIN
    CASE ID OF
        BEGIN
        LISTA;
        LISTALL;
```

```
        LISTDBASE;
        LISTC;
        LISTM(-1);
        CONTROL;
        ATTR;
        FA;
        FIND;
        FIND_IC;
        SWITCH;
        GC TC TERMINATE;
        CHANGE_KEY;
        SUPER_USER;
        DIAGNOSE(FALSE,TRUE);
        WRITE("COMMAND UNDEFINED.(FOR LIST OF CCMMANCS INPUT : LISTC)");
        END;
    END INTERPRET;

COMMENT PROCEDURES WHICH HANDLE CCMMANCS;

PROCEDURE LISTA;
COMMENT LIST ALL ATTRIBUTES OF TABLE "ATTRIBUTE";
    BEGIN
    INTEGER K;
    WRITE("LIST OF ATTRIBUTES WITHIN DATA BASE:");SPACE(1);
    WRITE("  IC            ATTRIBUTE");             SPACE(1);
    FOR K:=1 UNTIL NUMBER_OF_ATTRIBUTES DO WRITE(K,ATTRIBUTE(K));
    SPACE(1);
    END LISTA;

PROCEDURE LISTC;
COMMENT LIST AVAILABLE CCMMANDS;
    BEGIN
    INTEGER K;
    WRITE("LIST CF AVAILABLE COMMANDS:");
    WRITE("(FOR DETAILS SEE PROGRAM DESCRIPTION.)");
    SPACE(1);
    FOR K:=1 UNTIL NUMBER_CF_CCMMANDS-3 DO WRITE(K,":",COMMAND_STACK(K));
    SPACE(1);
    END LISTC;

PROCEDURE LISTDBASE;
COMMENT LIST ALL INFORMATION STORED IN DATA BASE;
    BEGIN
    INTEGER K;
```

```
    IF  FOUT  = 0 THEN
    WRITE("DATA BASE CONTAINS:");
    SPACE(2);
    FOR K:=1 UNTIL MEMBER DO
        BEGIN
        LISTM(K);
        IF  FOUT  = 0 THEN
        WRITE("......................................................");
        END;
    SPACE(2);
    END LISTDBASE;


PROCEDURE LISTALL;
COMMENT LIST ALL MEMBERS OF DATA BASE USING KEY ATTRIBUTES;
    BEGIN
    INTEGER K;
    WRITE("DATA BASE HAS THE FOLLOWING MEMBERS:");
    SPACE(1);
    FOR K:=1 UNTIL MEMBER DO LISTSHORT(K,KEY);
    SPACE(1);
    END LISTALL;


PROCEDURE LISTSHORT(INTEGER VALUE ID,N);
COMMENT LIST NTH ATTRIBUTE AND ITS VALUE OF MEMBER GIVEN BY ID;
    BEGIN
    INTEGER PA, PM;
    PM:=FINDMEMBER(ID);    COMMENT CURRENT_NODE SET ;
    IF CURRENT_NODE = NIL THEN ERROR(4,ID)
    ELSE
        BEGIN
        WRITE("MEMBER",ID,":");          WRITE(ATTRIBUTE(N));
        PA:=FINDATTRIBUTE(N);
        IF PA=NIL THEN WRITE("?")
        ELSE
        LISTATTRIBUTE(PA);
        END;
    END LISTSHORT;


INTEGER PROCEDURE FINDMEMBER(INTEGER VALUE ID);
COMMENT RETURN POINTER TO ID-NODE OF MEMBER OF GIVEN ID;
    BEGIN
    INTEGER NEXT, POINTER;
    NEXT:=FIRST_MEMBER;
    POINTER:=NIL;
    WHILE NEXT ¬= NIL DO
```

```
        BEGIN
        IF CDR2(NEXT+1) = ID THEN
            BEGIN
            POINTER:=NEXT;
            NEXT:=NIL;
            END
        ELSE NEXT:=CDR2(NEXT);
        END;
    CURRENT_NODE:=POINTER;
    POINTER
    END FINDMEMBER;

INTEGER PROCEDURE FINDATTRIBUTE(INTEGER VALUE N);
COMMENT RETURN POINTER TO ATTRIBUTE STRUCTURE SPECIFIED BY N.
    CURRENT_NODE POINTS TO MEMBER IN CONTROL;
    BEGIN
    INTEGER NEXT, POINTER;
    NEXT:=CAR2(CURRENT_NODE);
    POINTER:=NIL;
    WHILE NEXT ¬= NIL DO
        BEGIN
        IF CDR2(NEXT+1) =  N THEN
            BEGIN
            POINTER:=CAR2(NEXT);
            NEXT:=NIL;
            END
        ELSE NEXT:=CDR2(NEXT);
        END;
    POINTER
    END FINDATTRIBUTE;

PROCEDURE LISTATTRIBUTE(INTEGER VALUE P);
COMMENT LIST ATTRIBUTE VALUE POITED AT BY P;
    BEGIN
    STRING(80) BUFFER;
    STRING(4) CHAR;
    INTEGER I, X;
    LOGICAL DONE;
    DONE:=FALSE;
    BUFFER:=" ";
    X:=0;
    IF FOUT = 1 THEN
        BEGIN
        IF CAR2(P+1) = 1 THEN  WRITE("@");
        IF CAR2(P+1) = 2 THEN  WRITE("$");
        END;
```

```
WHILE DONE = FALSE DO
     BEGIN
     CHAR:=EXTRACT(P);
     FOR I:=0 UNTIL 3 DO
          BEGIN
          BUFFER(X|1):=CHAR(I|1);
          X:=X+1;
          IF CHAR = "#" THEN DONE:=TRUE;
          IF X>79 THEN
               BEGIN
               IF    FOUT     = 1 THEN WRITEON(BUFFER) ELSE
               WRITE(BUFFER);
               X:=0;
               BUFFER:=" ";
               END;
          END;
     P:=CDR2(P+1);
     IF P = NIL THEN DONE:=TRUE;
     END;
 IF X ¬= 0 THEN
     BEGIN
     IF FOUT = 1 THEN WRITEON(BUFFER) ELSE WRITE(BUFFER);
     END;
 END LISTATTRIBUTE;


PROCEDURE LISTM(INTEGER VALUE ID);
COMMENT LIST ALL ATTRIBUTES OF MEMBER TO BE SPECIFIED BY INPUT;
    BEGIN
    INTEGER NEXTA;
    IF ID<1 THEN
         BEGIN
         WRITE("INPUT ID:"); SPACE(1);
         READ(ID);
         END;
    NEXTA:=FINDMEMBER(ID);
    IF NEXTA = NIL THEN ERROR(4,ID)
    ELSE
         BEGIN
         SPACE(1);
         IF  FOUT  = 0 THEN
         WRITE("MEMPER",ID,":");
         NEXTA:=CAR2(NEXTA);
         WHILE NEXTA ¬= NIL DO
              BEGIN
              ID:=CDR2(NEXTA+1);
              SPACE(1);
              WRITE(ATTRIBUTE(ID));
```

```
            LISTATTRIBUTE(CAR2(NEXTA));
            NEXTA:=CDR2(NEXTA);
            END;
        END;
    IF      FOUT    = 1 THEN WRITE("#");
    END LISTM;


PROCEDURE CONTROL;
COMMENT FIND DESIRED MEMBER AND TAKE IT INTO CONTROL;
    BEGIN
    INTEGER ID;
    WRITE("INPUT ID:"); SPACE(1);
    READ(ID);
    CPOINTER:=FINDMEMBER(ID);
    IF CPOINTER = NIL THEN ERROR(4,ID);
    END CONTROL;


PROCEDURE ATTR;
COMMENT LIST SPECIFIED ATTRIBUTE OF MEMBER IN CONTROL;
    BEGIN
    INTEGER N, POINTER;
    WRITE("INPUT ATTRIBUTE ID:"); SPACE(1);
    READ(N);
    CURRENT_NODE:=CPOINTER;
    POINTER:=FINDATTRIBUTE(N);
    IF POINTER = NIL THEN WRITE("ATTRIBUTE NOT FOUND.")
    ELSE
        BEGIN
        WRITE(ATTRIBUTE(N));
        LISTATTRIBUTE(POINTER);
        END;
    SPACE(1);
    END ATTR;


PROCEDURE FA;
COMMENT FIND ATTRIBUTE OF MEMBER IN CONTROL SPECIFIED BY INPUT STRING;
    BEGIN
    STRING(80) A;
    STRING(64) B;
    INTEGER N, POINTER;
    WRITE("INPUT ATTRIBUTE TEXT:"); SPACE(1);
    READCARD(A);
    B:=A(0|64);
    N:=FIND_ATTRIBUTE(B,64);
    IF N = 0 THEN WRITE("ATTRIBUTE NOT LISTED, REPEAT.")
```

```
ELSE
    BEGIN
    CURRENT_NODE:=POINTER;
    POINTER:=FINDATTRIBUTE(N);  COMMENT FIND ATTRIBUTE WITHIN
                                        LINKED LIST;
    IF POINTER = NIL THEN WRITE("ATTRIBUTE NOT FOUND.")
    ELSE LISTATTRIBUTE(POINTER);
    END;
SPACE(1);
END FA;


PROCEDURE FIND;
COMMENT FIND ALL MEMBERS MEETING SPECIFIED CONDITIONS;
    BEGIN
    INTEGER NEXT, TEMP;
    NUMBER_OF_CONDITIONS:=0;
    GET_CONDITIONS;
    IF NUMBER_OF_CONDITIONS > 0 THEN
        BEGIN
        NEXT:=LIST_POINTER;
        WHILE NEXT ¬= NIL DO
            BEGIN
            TEMP:=CDR1(NEXT);
            FREE1(NEXT);
            NEXT:=TEMP;
            END;
        LIST_POINTER:=NIL;
        TEST_ALL;
        IF LIST_POINTER ¬= NIL THEN WRITE_RESULTS
        ELSE WRITE("NO MATCH.");
        END;
    END FIND;


PROCEDURE FIND_ID;
COMMENT  CALLS FIND AND SETS TOGGLE SUCH THAT ONLY KEY ATTRIBUTES ARE
         LISTED;
    BEGIN
    TOGGLE:=1;
    FIND;
    TOGGLE:=0;
    END FIND_ID;


PROCEDURE GET_CONDITIONS;
COMMENT GET ATTRIBUTES AND CORRESPONDING VALUES FOR FIND FUNCTION;
    BEGIN
```

```
    LOGICAL DONE;
    DONE:=FALSE;
    WHILE DONE = FALSE DO
         BEGIN
         DONE:=GET_REQUEST;
         IF DONE = FALSE THEN GET_QVALUE;
         END;
    END GET_CONDITIONS;


LOGICAL PROCEDURE GET_REQUEST;
COMMENT GET NEXT REQUEST OF FIND FUNCTION;
    BEGIN
    INTEGER J;
    STRING(80) BUFFER;
    STRING(64) B;
    LOGICAL LEGAL, DONE;
    DONE:=FALSE;
    LEGAL:=FALSE;
    WHILE ((LEGAL=FALSE) AND (DONE=FALSE)) DO
         BEGIN
         WRITE("INPUT ATTRIBUTE OR Q:");          SPACE(1);
         READCARD(BUFFER);
         IF BUFFER = "Q" THEN
              BEGIN
              LEGAL:=TRUE;
              NUMBER_OF_CONDITIONS:=NUMBER_OF_CONDITIONS + 1;
              REQUEST(NUMBER_OF_CONDITIONS):="Q";
              DONE:=TRUE;
              END
         ELSE
              BEGIN
              B:=BUFFER(0|64);
              J:=FIND_ATTRIBUTE(B,64);
              IF J ¬= 0 THEN
                   BEGIN
                   LEGAL:=TRUE;
                   NUMBER_OF_CONDITIONS:=NUMBER_OF_CONDITIONS + 1;
                   REQUEST(NUMBER_OF_CONDITIONS):=B;
                   END
              ELSE WRITE("ATTRIBUTE NOT FOUND.");
              END;
         IF ((NUMBER_OF_CONDITIONS = 5) AND (REQUEST(5) ¬= "Q")) THEN
              BEGIN
              WRITE("ILLEGAL REQUEST.");
              NUMBER_OF_CONDITIONS:=0;
              DONE:=TRUE;  COMMENT COMMAND NOT EXECUTED;
              END;
```

```
        END;
    DONE
    END REQUEST;

PROCEDURE GET_QVALUE;
COMMENT GET QVALUE FOR CURRENT ATTRIBUTE USED IN FIND FUNCTION;
    BEGIN
    STRING(80) BUFFER;
    WRITE("INPUT VALUE  OF ATTRIBUTE(USE CORRECT FORMAT):"); SPACE(1);
    READCARD(BUFFER);
    QVALUE(NUMBER_OF_CONDITIONS):=BUFFER(0|64);
    SPACE(1);
    END GET_QVALUE;

LOGICAL PROCEDURE MATCHING(INTEGER VALUE I);
COMMENT RETURNS TRUTH VALUE OF ITH CONDITION OF MEMBER IDENTIFIED BY
        CURRENT_NODE;
    BEGIN
    INTEGER N, J;
    LOGICAL TRUTH;
    N:=FIND_VALUE_LENGTH(I);
    J:=FIND_ATTRIBUTE(REQUEST(I),64);
    CURRENT_ID:=J;
    IF J=0 THEN TRUTH:=FALSE
    ELSE TRUTH:=EQUAL(I,N);
    TRUTH
    END MATCHING;

INTEGER PROCEDURE FIND_VALUE_LENGTH(INTEGER VALUE I);
    COMMENT RETURN LENGTH - 1 OF QVALUE(I);
    BEGIN
    INTEGER L;
    STRING(64) Q;
    L:=63;
    Q:=QVALUE(I);
    WHILE Q(L|1) = " " DO
        BEGIN
        L:=L-1;
        IF ((L=0) AND (Q(0|1) = " ")) THEN Q(0|1):="?";
        END;
    VALUE_LENGTH(I):=L;
    L
    END FIND_VALUE_LENGTH;
```

```
LOGICAL PROCEDURE EQUAL(INTEGER VALUE I,N);
COMMENT RETURN TRUTH VALUE FOR: QVALUE(I) = ITH ATTRIBUTE VALUE OF
        MEMBER BEING EXAMINED.
        CURRENT_NODE IDENTIFIES MEMBER.
        CURRENT_ID IDENTIFIES ATTRIBUTE BEING COMPARED;
    BEGIN
    INTEGER J, POINTER, TYPE;
    STRING(64) VAL1, VAL2;
    STRING(64) U, L;
    LOGICAL TRUTH;
    TRUTH :=TRUE;
    J:=0;
    VAL1:=QVALUE(I);
    VAL2:=" ";
    POINTER:=FINDATTRIBUTE(CURRENT_ID);
    IF POINTER = NIL THEN TRUTH:=FALSE
    ELSE
        BEGIN
        TYPE:=CAR2(POINTER+1); COMMENT GET VALUE TYPE;
        WHILE ((POINTER ¬= NIL) AND (J < 64)) DO
            BEGIN
            VAL2(J|4):=EXTRACT(POINTER);
            J:=J+4;
            POINTER:=CDR2(POINTER+1);
            END;
        IF TYPE = 0 THEN
            BEGIN
            COMMENT SINGLE DISCRETE VALUE;
            IF VAL2(0|3)¬="***" THEN
                BEGIN
                J:=0;
                WHILE ((TRUTH = TRUE) AND (J < N+1)) DO
                    BEGIN
                    IF VAL1(J|1) ¬= VAL2(J|1) THEN TRUTH:=FALSE;
                    J:=J+1;
                    END;
                IF ((VAL2(J|1) ¬="#") AND (VAL2(J|1) ¬=" ")) THEN
                                                           TRUTH:=FALSE;
                END;
            END;
        IF TYPE = 1 THEN
            BEGIN
            COMMENT RANGE VALUE;
            IF VAL1(0|1) = "+" THEN
                BEGIN
                VAL1(0|64):=VAL1(1|63);
                VAL1(63|1):=" ";
                END;
```

```
            L:=LLIMIT(VAL2);
            IF L = "?" THEN TRUTH:=FALSE
            ELSE
                BEGIN
                U:=ULIMIT(VAL2);
                SHIFT(U);
                SHIFT(L);
                SHIFT(VAL1);
                TRUTH:=COMPARE_RANGE(U,L,VAL1);
                END;
            END;
        IF TYPE =2 THEN
            BEGIN
            COMMENT MORE THAN ONE VALUE;
            TRUTH:=COMPARE_MANY(VAL1,VAL2,N);
            END;

        END;
    TRUTH
    END EQUAL;


PROCEDURE TEST_CONDITIONS;
COMMENT TEST ALL CONDITIONS FOR MEMBER IDENTIFIED BY CURRENT_NODE;
COMMENT IF ALL CONDITIONS HOLD ADD MEMBER TO LINKED LIST;
    BEGIN
    INTEGER A, I;
    LOGICAL FAIR;
    FAIR:=TRUE;
    I:=1;
    WHILE ((REQUEST(I) ¬= "Q") AND ( FAIR = TRUE )) DO
        BEGIN
        FAIR:=MATCHING(I);
        I:=I+1;
        END;
    IF FAIR = TRUE THEN
        BEGIN
        A:=ALLOCATE1;
        IF LIST_POINTER = NIL THEN LIST_POINTER:=A
        ELSE SETCDR1(LAST,A);
        LAST:=A;
        SETCDR1(LAST,NIL);
        SETCAR1(LAST,CDR2(CURRENT_NODE+1));
        END;
    END TEST_CONDITIONS;


PROCEDURE TEST_ALL;
```

```
COMMENT TEST ALL DATA BASE MEMBERS FOR ALL CONDITIONS REQUESTED;
    BEGIN
    INTEGER POINTER;
    POINTER:=FIRST_MEMBER;
    WHILE POINTER ¬= NIL DO
        BEGIN
        CURRENT_NODE:=POINTER;
        TEST_CONDITIONS;
        POINTER:=CDR2(POINTER);
        END;
    END TEST_ALL;

PROCEDURE LIST_MEMBER(INTEGER VALUE ID);
COMMENT LIST ALL INFORMATION OF MEMBER SPECIFIED BY ID;
    LISTM(ID);


PROCEDURE WRITE_RESULTS;
COMMENT WRITE RESULTS OF FIND OR FIND_ID FUNCTION;
    BEGIN
    INTEGER POINTER, ID;
    WRITE("LIST OF DATA BASE MEMBERS MATCHING GIVEN CONDITIONS:");
    WRITE("************************************************************");
    SPACE(1);
    POINTER:=LIST_POINTER;
    WHILE POINTER ¬= NIL DO
        BEGIN
        ID:=CAR1(POINTER);
        IF TOGGLE = 0 THEN LIST_MEMBER(ID)
        ELSE LISTSHORT(ID,KEY);
        POINTER:=CDR1(POINTER);
        END;
    WRITE("************************************************************");
    SPACE(1);
    END WRITE_RESULTS;



LOGICAL PROCEDURE COMPARE_MANY(STRING(64) VALUE V1,V2;INTEGER VALUE N);
COMMENT RETURN TRUTH VALUE WHETHER OR NOT ONE OF THE VALUES OF V2 MATCH
         WITH STRING IN V1;
    BEGIN
    STRING(64) VC;
    LOGICAL MATCH;
    INTEGER I, J, K;
    MATCH:=FALSE;
```

```
    I:=C;
    J:=C;
    K:=C;
    VC:=" ";
    WHILE ((MATCH = FALSE) AND (J < 64)) DO
        BEGIN
        IF ((V2(J|1) = ",") OR (V2(J|1) = "#")) THEN
            BEGIN
            MATCH:=(VC=V1);
            K:=0;
            VC:=" ";
            IF  V2(J|1)  = "#" THEN J:=64;
            I:=J+1;
            J:=I;
            END
        ELSE
            BEGIN
            VC(K|1):=V2(J|1);
            J:=J+1;
            K:=K+1;
            END;
        END;
    MATCH
    END COMPARE_MANY;


STRING(64) PROCEDURE LLIMIT(STRING(64) VALUE VAL);
COMMENT RETURN LOWER LIMIT OF RANGE VALUE;
    BEGIN
    STRING(64) L;
    INTEGER I;
    L:=" ";
    I:=C;
    WHILE VAL(I|4) ¬= " TO " DO
        BEGIN
        IF VAL(I|1) ¬="+" THEN
        L(I|1):=VAL(I|1);
        I:=I+1;
        IF I > 63 THEN
            BEGIN
            WRITE("RANGE VALUE FORMAT OF MEMBER",CDR2(CURRENT_NODE+1),
                  "HAS WRONG FORMAT.");
            L:="?";
            END;
        END;
    L
    END LLIMIT;
```

```
STRING(64) PROCEDURE ULIMIT(STRING(64) VALUE VAL);
COMMENT RETURN UPPER LIMIT OF RANGE VALUE;
    BEGIN
    STRING(64) U;
    INTEGER I, J;
    U:=" ";
    I:=0;
    J:=0;
    WHILE VAL(I|4) ¬= " TO " DO I:=I+1;
    I:=I+4;
    WHILE ((VAL(I|1) ¬= "#" ) AND ( I < 64)) DO
        BEGIN
        IF VAL(I|1) ="+" THEN J:=J-1 ELSE
        U(J|1):=VAL(I|1);
        I:=I+1;
        J:=J+1;
        IF I > 63 THEN WRITE("MISSING #.",
                            "CHECK MEMBER:",CDR2(CURRENT_NODE+1));
        END;
    U
    END ULIMIT;



PROCEDURE CHANGE_KEY;
COMMENT GIVES USER THE CHANCE TO REDEFINE KEY ATTRIBUTE;
    BEGIN
    INTEGER K;
    WRITE("OLD KEY ATTRIBUTE:",KEY );
    WRITECN("(",ATTRIBUTE(KEY),")");
    WRITE("INPUT NEW KEY ID(INTEGER):");
    SPACE(1);
    READ(K);
    IF ((K<1) OR (K>NUMBER_OF_ATTRIBUTES)) THEN
        WRITE("INVALID INPUT. KEY NOT CHANGED.")
    ELSE
        BEGIN
        KEY:=K;
        WRITE("NEW KEY ATTRIBUTE:",KEY);
    WRITECN("(",ATTRIBUTE(KEY),")");       SPACE(1);
        END;
    SPACE(1);
    END CHANGE_KEY;

PROCEDURE SUPER_USER;
```

```
COMMENT DESIGNED FOR EXPERIMENTS AND TESTING;
    BEGIN
    STRING(80) BUFFER;
    INTEGER ADDRESS;
    WRITE("BEGIN OF SUPER USER MODE.");
    WRITE("DO YOU WANT TO WRITE THE DATA BASE ON FILE?");
    WRITEON("(T/F)");                  WRITE(" ");
    READCARD(BUFFER);
    IF BUFFER = "T" THEN
         BEGIN
         WRITE(" OUTPUT TO FILE"); WRITE(" ");
         PRINTOUT;
         POUT:=1;
         LISTDBASE;
         WRITEON("#");
         TYPOUT;
         POUT:=0;
         WRITE(" OUTPUT TO TERMINAL"); WRITE(" ");
         END;
    WRITE("DO YOU WANT TO ADD NEW MEMBERS TO THE DATA BASE?");
    WRITEON("(T/F)");                  WRITE(" ");
    READCARD(BUFFER);
    IF BUFFER = "T" THEN
         BEGIN
         STORE;
         END;
    WRITE("DO YOU WANT TO EXAMINE DATA STORAGE?");
    WRITEON("(T/F)");                  WRITE(" ");
    READCARD(BUFFER);
    IF BUFFER = "T" THEN
         BEGIN
         WRITE("INPUT ADDRESS.(POSITVE INTEGER < 40,000)");
         WRITE("TO QUIT INPUT NEGATIVE INTEGER.");
         SPACE(1);
         READ(ADDRESS);
         WHILE ADDRESS > -1 DO
              BEGIN
              WRITE(ADDRESS+1,":",CAR2(ADDRESS+1),CDR2(ADDRESS+1));
              WRITE(ADDRESS,":",CAR2(ADDRESS),CDR2(ADDRESS),
                   EXTRACT(ADDRESS));
              WRITE("NEXT ADDRESS:");
              SPACE(1);
              READ(ADDRESS);
              END;
         END;
    WRITE("END OF SUPER USER MODE.");
    SPACE(1);
    END SUPER_USER;
```

```
PROCEDURE SWITCH;
COMMENT CHANGE OUTPUT FROM TERMINAL TO FILE AND VICE VERSA;
    BEGIN
    IF MAINSWITCH = 0 THEN
        BEGIN
        MAINSWITCH:=1;
        WRITE(" OUTPUT TO FILE"); WRITE(" ");
        PRINTOUT;
        END
    ELSE
        BEGIN
        MAINSWITCH:=0;
        TYPOUT;
        WRITE(" OUTPUT TO TERMINAL"); WRITE(" ");
        END;
    END SWITCH;


LOGICAL PROCEDURE COMPARE_RANGE(STRING(64) VALUE U, L, VAL1);
COMMENT RETURN TRUTH VALUE WETHER OR NOT VAL1 LIES IN RANGE OF U AND L;
    BEGIN
    LOGICAL TRUTH;
    IF U(0|1) = "-" THEN
        BEGIN
        IF VAL1(0|1) = "-" THEN TRUTH:=(VAL1 >= U)
        ELSE TRUTH:=FALSE;
        END
    ELSE
        BEGIN
        IF VAL1(0|1) ="-" THEN  TRUTH :=TRUE
        ELSE TRUTH:=(U>=VAL1);
        END;
    IF L(0|1) ="-" THEN
        BEGIN
        IF VAL1(0|1) ="-" THEN TRUTH:=((TRUTH) AND (L>=VAL1));
        END
    ELSE
        BEGIN
        IF VAL1(0|1) ="-" THEN TRUTH:=FALSE
        ELSE TRUTH:=((TRUTH) AND (VAL1>=L));
        END;
    TRUTH
    END COMPARE_RANGE;


PROCEDURE SHIFT(STRING(64) VALUE RESULT S);
COMMENT  ADJUST RANGE VALUES TO THE RIGHT;
```

```
BEGIN
STRING(64) NEW;
INTEGER I, J, K;
K:=31;
NEW:=" ";
IF S(0|1) ="-" THEN
     BEGIN
     NEW(0|1):="-";
     I:=1;
     END
ELSE I:=0;
J:=I;
WHILE S(J|1) ¬= " " DO J:=J+1;
J:=J-1;
WHILE J>=I DO
     BEGIN
     NEW(K|1):=S(J|1);
     K:=K-1;
     J:=J-1;
     END;
S:=NEW;
END SHIFT;

COMMENT MAIN;
INITIALIZE_ALL;
GET_OLD_FILE_INFORMATION;
TYPIN;
GET_COMMANDS;
OVERFLOW:
TERMINATE:
WRITE("END OF PROGRAM.");
END.
```

WORKSHEET FOR CODING PHASE OF PROJECT # : 4

Beginning of Coding (day/time) : 4/20/1400

End of Coding (day/time) : 5/02/1200

Man hours : 24.5 (including punching of cards)

CODING BEGIN DAY/TIME	END DAY/TIME	PROGRAM PART	1) ERROR #	DAY TIME	COMMENT
					1) Record when error is detected.
04/20/1400	04/20/1800	Coding of primitives and utilities			A great number of primitives could be copied from project #2.
04/20/2030	04/20/2330	Punching cards			
04/21/1330	04/21/1530	Coding of command functions			
04/21/1530	04/21/1830	Punching cards	6	1830	C9
04/21/2030	04/21/2130	Coding of command functions and punching cards			
04/21/2230	04/21/2330	Coding of command functions and punching cards			
04/22/1500	04/22/1800	Coding of retrieval functions			
04/22/1830	04/22/1930	Punching cards			
04/25/1330	04/25/1630	Coding of re-retrieval functions for multiple values and range values			

WORKSHEET FOR CODING PHASE OF PROJECT # : 4

Beginning of Coding (day/time) : 4/20/1400

End of Coding (day/time) : 5/02/1200

Man hours : 24.5 (including punching of cards)

CODING BEGIN DAY/TIME	END DAY/TIME	PROGRAM PART	ERROR # [1]	DAY TIME	COMMENT
					1) Record when error is detected.
04/25/2100		Punching cards			
			35	2100	A1
			36	2200	A1
	04/25/2200				
04/26/1530	04/26/1630	Coding of debugging aids and punching cards			
04/27/2030	04/27/2100	Changes according to design review	48	2050	C17
05/02/1100	05/02/1200	Coding of new procedures to handle range value comparison and punching cards			Change due to error # 50.

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 4 DEBUG Run # : 1

Begin of Debug Run (day/time) : 04/20/2330

End of Debug Run (day/time) : 04/21/2030

# of Debug Steps incl. in Debug Run: 4 CPU time for Debug run (sec): 0.0

CPU time for necessary compiles (sec) : 19.39

a) 3.93 b) 4.42 c) 5.34 d) 5.70 e) f) g)

Man hours for this Debug Run : 2.5 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	Primi- tives and utili- ties	Get error free compile	5 compile errors	4/20 2330 4/21 1030 4/21 1100	1.0	1 2 3 4 5	C6 A1 A1 A1 A1
2	Primi- tives, utili- ties, input funct- ions	Get error free compile	1 compile error	4/21 1830 1900 4/21 1930	0.5	7	C17
3		repeat step 2	1 compile error	4/21 1930 2000 4/21 2000	0.5	8	C12
4		repeat step 2	O.K.	4/21 2000 4/21 2030	0.5		

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 4 DEBUG Run # : 2

Begin of Debug Run (day/time) : 04/21/2130

End of Debug Run (day/time) : 04/22/1830

# of Debug Steps incl. in Debug Run: 6 CPU time for Debug run (sec): 14.83

CPU time for necessary compiles (sec) : 39.55

a) 6.48 b) 5.52 c) 5.68 d) 4.47 e) 8.26 f) 9.14 g)

Man hours for this Debug Run : 5.5 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when
1	All	Get error free compile	2 compile	4/21			error
	except		errors	2130	0.5		occurs
	retrie-			2130		9	C7
	val					10	A1
	funct-			4/21			
	ions			2200			
2		repeat step 1 and check	1 program	4/21			
		initialization	error	2200	0.5		
				2200		11	D7
				4/21			
				2230			
3		repeat step 2	5 program	4/21			
			errors	2330	2.5		
				4/22			
				1000		12	C28
				1020		13	C21
				1030		14	C21
				1100		15	C28
				1110		16	D12
				4/22			
				1200			
4	All	Get error free compile	2 compile	4/22			
	except	(including command	errors	1200	0.5		
	retrie-	functions)		1200		17	C6
	val			1230		18	C17
	funct-			4/22			
	ions			1230			
5		repeat step 4 and check	2 program	4/22			
		initialization	errors	1400	1.0		
				1430		19	D12
				4/22			
				1500		20	C27
6		repeat step 5	O.K.	4/22			
				1800	0.5		

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 4 DEBUG Run # : 3

Begin of Debug Run (day/time) : 04/22/1830

End of Debug Run (day/time) : 04/22/2200

# of Debug Steps incl. in Debug Run: 2 CPU time for Debug run (sec): 11.93

CPU time for necessary compiles (sec) : 17.75

a) 8.71 b) 9.04 c) d) e) f) g)

Man hours for this Debug Run : 2.5 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	1) ERROR #	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	All except functions to compare range values or multiple discrete values	Get error free compile	3 compile errors	4/22 1930	0.5		
				1945		21	A1
				1950		22	A2
				1955		23	C23
				4/22 2000			
2		repeat step 1 and examine storage of data base information (all attributes and values should be implemented according to design)	O.K.	4/22 2000			
				4/22 2200			

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 4 DEBUG Run # : 4

Begin of Debug Run (day/time) : 04/22/2200

End of Debug Run (day/time) : 04/25/1100

# of Debug Steps incl. in Debug Run: 6 CPU time for Debug run (sec): 43.47

CPU time for necessary compiles (sec) : 73.14

a) 11.36 b) 9.73 c) 9.84 d) 9.96 e) 10.87 f) 10.49 g) 10.89

Man hours for this Debug Run : 7.5 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	1) ERROR #	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	All except compa-rison of mul-tiple values or range values	Check basic functions (except FIND function)	1 program error	4/22 2200	0.5		
				2200		24	D12
				4/22 2230			
2		repeat step 1	1 program error	4/22 2230	3.0		
				4/23 1230		25	D3
3		repeat step 1	1 program error	4/23 1230	1.0		
				1300		26	D13
				4/23 1330			
4		repeat step 1	1 program error	4/23 1330	1.0		
				1400		27	D15
				4/23 1430			
5		repeat step 1	2 program errors	4/23 1430	1.0		
				1500		28	D12
				1500		29	C27
				4/23 1530			
6		repeat step 1	O.K.	4/24 1700	1.0		
				4/25 1100			

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 4 DEBUG Run # : 5

Begin of Debug Run (day/time) : 04/25/1630

End of Debug Run (day/time) : 04/25/2400

# of Debug Steps incl. in Debug Run: 5 CPU time for Debug run (sec): 8.95

CPU time for necessary compiles (sec) : 51.3

a) 11.02 b) 11.48 c) 11.90 d) 9.37 e) 7.53 f) g)

Man hours for this Debug Run : 5.5 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	Find funct-tion (single dis-crete values)	Check FIND function for trivial cases Check input of conditions (0-4 conditions should be stacked according to design)	3 program errors	4/25 1630 1630 1700 1730 4/25 1730	1.0	30 31 32	A1 C27 D12
2		repeat step 1 (using trace)	1 program error	4/25 1700 2030 4/25 2030	2.0	33	B4 (Error #32 not corrected properly)
3		repeat step 1	1 program error	4/25 2030 2100 4/25 2100	0.5	34	C28
4		repeat step 1	1 program error	4/25 2200 2230 4/25 2245	0.5	37	
5		repeat step 1	O.K.	4/25 2245 4/25 2400	1.5		

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 4   DEBUG Run # : 6

Begin of Debug Run (day/time) : 04/25/2400

End of Debug Run (day/time) : 04/26/1930

# of Debug Steps incl. in Debug Run: 7   CPU time for Debug run (sec): 10.23

CPU time for necessary compiles (sec) : 67.84

a) 6.57   b) 12.88   c) 12.66   d) 6.66   e) 8.96   f) 10.27   g) 9.84

Man hours for this Debug Run : 8.0 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when
1	A11	Get error free compile	3 compile	4/25			error
		and test FIND function	errors	2400	2.0		occurs
		for trivial cases		4/16			
		(all members listed must		0900		38	A1
		match conditions speci-		0910		39	A1
		fied by input)		0915		40	A1
				4/26			
				1000			
2		repeat step 1	2 program	4/26			
			errors	1000	2.0		
				1050		41	C28
				1100		42	C28
				4/26			
				1200			
3		repeat step 1	1 program	4/26	1.0		
			error	1330		43	A1
				4/26			
				1440			
4		repeat step 1	1 compile	4/26			
		(including debugging	error	1630	1.0		
		aids)		4/26			
				1730		44	A1
5		repeat step 4	1 program	4/26			
			error	1730	0.5		
				1745		45	A3
				4/26			
				1800			
6		repeat step 4	1 program	4/26			
			error	1800	0.5		
				1815		46	A1
				4/26			
				1830			
7		repeat step 4	O.K.	4/26			
				1830	1.0		

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 4 DEBUG Run # : 7

Begin of Debug Run (day/time) : 04/26/2000

End of Debug Run (day/time) : 04/26/2300

# of Debug Steps incl. in Debug Run: 2 CPU time for Debug run (sec): 7.0

CPU time for necessary compiles (sec) : 18.14

a) 9.73 b) 8.41 c) d) e) f) g)

Man hours for this Debug Run : 3.0 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	All	Test FIND function for various input (including also extreme conditions) (all queries should be answered according design)	1 program error	4/26 2000 2120 4/26 2200	2.0	47	C28
2		repeat step 1	O.K.	4/26 2200 4/26 2300	1.0		

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 4 DEBUG Run # : 8

Begin of Debug Run (day/time) : 04/27/1500

End of Debug Run (day/time) : 04/27/1900

# of Debug Steps incl. in Debug Run: 1 CPU time for Debug run (sec): 11.98

CPU time for necessary compiles (sec) : 10.02

a) 10.02 b) c) d) e) f) g)

Man hours for this Debug Run : 3.0 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	All	Repeat Debug Run #7 under CP/CMS (same results expected)	O.K.	4/27 1500 4/27 1900	3.0		

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 4 DEBUG Run # : 9

Begin of Debug Run (day/time) : 04/28/1400

End of Debug Run (day/time) : 04/28/1600

# of Debug Steps incl. in Debug Run: 1 CPU time for Debug run (sec): 19.58

CPU time for necessary compiles (sec) : 9.46

a) 9.46 b) c) d) e) f) g)

Man hours for this Debug Run : 2.0 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	1) ERROR #	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	All	Various tests under CP/CMS: -Test initialization -Test LISTA (list of all attributes) -Test LISTC (list all available commands) -Test LISTM (check for all members and illegal input) -Check program for input of undefined commands -Test CONTROL (check for illegal input also) -Test functions FA and ATTR for several members in control -Test SWITCH (examine out-put file) -Test KEY (change of key attribute as desired by user) -Test LISTDBASE (all mem-bers are listed with all all attribute value pairs) -Test LISTALL (members are listed only by key attri-bute) -Test FIND function (using various input combi-nations)	O.K.	4/28 1400 4/28 1600	2.0		

WORKSHEET FOR DEBUGGING PHASE

PROJECT # : 4 DEBUG Run # : 10

Begin of Debug Run (day/time) : 04/28/1600

End of Debug Run (day/time) : 04/28/1800

# of Debug Steps incl. in Debug Run: 1 CPU time for Debug run (sec): 12.08

CPU time for necessary compiles (sec) : 9.46

a) 9.46 b) c) d) e) f) g)

Man hours for this Debug Run : 2.0 (including preparation of debug run)

STEP #	PROGRAM PART	OBJECTIVE AND EXPECTED RESULT	ACTUAL RESULT	DAY TIME	MAN HOURS /STEP	ERROR # 1)	COMMENTS AND CODED ERROR TYPES
							1) Record when error occurs
1	All	Various tests under CP/CMS:		4/28 1600	2.0		
		a) Test SU (super user function)	1 program error	1620		49	C27
		b) Test FIND function for 0 conditions (all members should be listed)	O.K.				
		c) Test FIND amd FIND ID using 1-4 different conditions (check correctness of program especially multiple values and range values)	O.K.				
		d) Test KEY function for illegal inputs (program should provide an appropriate error message)	O.K.				
				4/28 1830			

ERROR LISTING

PROJECT # : 4

Begin of Project (day/time) : 04/19/1400

End of Project (day/time) : 04/03/1800

Man hours for total project : 101.0

ERROR #	PHASE in which ERROR was dis-covered	PHASE in which ERROR was made	ERROR TYPE (see ANNEX F)	TIME spent to solve the ERROR (Man min.)	# of OTHER STATEMENTS OR PARTS OF THE PROGRAM AFFECTED
1	Debugging	Coding	C6	5	
2	Debugging	Coding	A1	5	
3	Debugging	Coding	A1	5	
4	Debugging	Coding	A1	5	
5	Debugging	Coding	A1	5	
6	Coding	Coding	C9	5	
7	Debugging	Coding	C17	5	
8	Debugging	Coding	C12	5	
9	Debugging	Coding	C7	5	
10	Debugging	Coding	A1	5	
11	Debugging	Design	D7	5	
12	Debugging	Coding	C28	5	
13	Debugging	Coding	C21	15	
14	Debugging	Coding	C21	15	
15	Debugging	Coding	C28	10	
16	Debugging	Design	D12	25	
17	Debugging	Coding	C6	5	
18	Debugging	Coding	C17	5	
19	Debugging	Design	D12	30	
20	Debugging	Coding	C27	10	
21	Debugging	Coding	A1	5	
22	Debugging	Coding	A2	5	
23	Debugging	Coding	C23	5	
24	Debugging	Design	D12	10	
25	Debugging	Design	D3	15	
26	Debugging	Design	D13	30	
27	Debugging	Design	D15	20	
28	Debugging	Design	D12	30	
29	Debugging	Coding	C27	5	
30	Debugging	Coding	A1	5	

ERROR LISTING

PROJECT # : 4

Begin of Project (day/time) : 04/19/1400

End of Project (day/time) : 04/03/1800

Man hours for total project : 101.0

ERROR #	PHASE in which ERROR was dis-covered	PHASE in which ERROR was made	ERROR TYPE (see ANNEX F)	TIME spent to solve the ERROR (Man min.)	# of OTHER STATEMENTS OR PARTS OF THE PROGRAM AFFECTED
31	Debugging	Coding	C27	10	
32	Debugging	Design	D12	90	
33	Debugging	Debugging	B4	15	
34	Debugging	Coding	C28	15	
35	Coding	Coding	A1	5	
36	Coding	Coding	A1	5	
37	Debugging	Debugging	A1	5	
38	Debugging	Coding	A1	5	
39	Debugging	Coding	A1	5	
40	Debugging	Coding	A1	5	
41	Debugging	Coding	C28	5	
42	Debugging	Coding	C28	5	
43	Debugging	Coding	A1	5	
44	Debugging	Coding	A1	5	
45	Debugging	Debugging	A3	5	
46	Debugging	Coding	A1	10	
47	Debugging	Coding	C28	10	
48	Coding	Coding	C17	5	
49	Debugging	Coding	C27	30	
50	Testing	Design	D3	120	

ERROR LISTING (COMMENTS)

ERROR #	DAY TIME	COMMENTS (EVIDENCE, THOUGHTS, WHY WAS THE ERROR MADE? WHY AND HOW WAS THE ERROR DISCOVERED? ERROR BLOCKING, etc.)
	04/21	
1	1030	Errors #1 thru 5 are due to influence of fatigue.
2	1030	
3	1030	
4	1030	
5	1030	
6	1830	Found while checking previously written code during punching of cards.
7	1900	Different variable name assumed.
8	2000	Missing mandatory declaration.
9	2130	
10	2130	
11	2200	
	04/22	
12	1000	
13	1020	
14	1030	Errors #14 thru 16 are due to insufficient desk checking.
15	1100	
16	1110	
17	1200	
18	1230	
19	1430	
20	1500	
21	1945	
22	1950	Upper case key pressed while punching cards.
23	1955	Usage of "THEN" instead of "DO".(wrong format)
24	2200	
	04/23	
25	1230	
26	1300	
27	1400	
28	1500	
29	1500	
	04/25	
30	1630	

ERROR LISTING (COMMENTS)

ERROR #	DAY TIME	COMMENTS (EVIDENCE, THOUGHTS, WHY WAS THE ERROR MADE? WHY AND HOW WAS THE ERROR DISCOVERED? ERROR BLOCKING, etc.)
	04/25	
31	1700	
32	1730	No proper desk checking. (Error could have been avoided by desk checking.)
33	2030	Inappropriate correction of error #32. Debugging results were not interpreted correctly.
34	2100	Forgotten "+1".
35	2100	Errors #35 and 36 are due to fatigue.(trivial errors)
36	2200	
37	2230	
	04/26	
38	0900	Errors #38 thru 40 are due to lack of concentration while punching cards.
39	0910	
40	0915	
41	1050	
42	1100	
43	1330	
44	1730	
45	1745	
46	1815	
47	2120	Double negation in connection with logical AND could have been tested by usage of a truth table. The error could have been avoided this way during a desk test.
	04/27	
48	2050	Found while punching cards.
	04/28	
49	1620	
	05/01	
50	1700	It was surprising to the experiment programmer that using the EBCDIC character set the following comparison will be evaluated "TRUE": " - " > " + " Error # 50 was made because this relationship was not known.

PROJECT # 4

FINAL STATISTICS

Project name : DATA RETRIEVAL SYSTEM

Short description:

The program is designed for usage under CP/CMS. It expects an input file labeled "DBASE INPUT" to contain data base information in a particular format such that the program can read all information and store it into memory. After all data base members are defined by input and implemented within linked lists the user may operate upon the data base using functions from a previously defined set of functions.

The program has been designed and implemented in such a manner that it will be easy to implement more functions or to extent the currently defined limits of the program.

Quantitative measures:

1. # of source statements : 1084

2. Total man hours for project : 101.0

3. Man hours spent in

   a) Design : 24.0

   b) Coding : 24.5

   c) Debugging : 41.5

   d) Testing : 11.0

4. CPU time for compiles: 343.57 sec.

5. CPU time for debug runs: 140.05 sec.

6. CPU time for test runs: 38.04 sec.

7. # of test and debug runs: 13

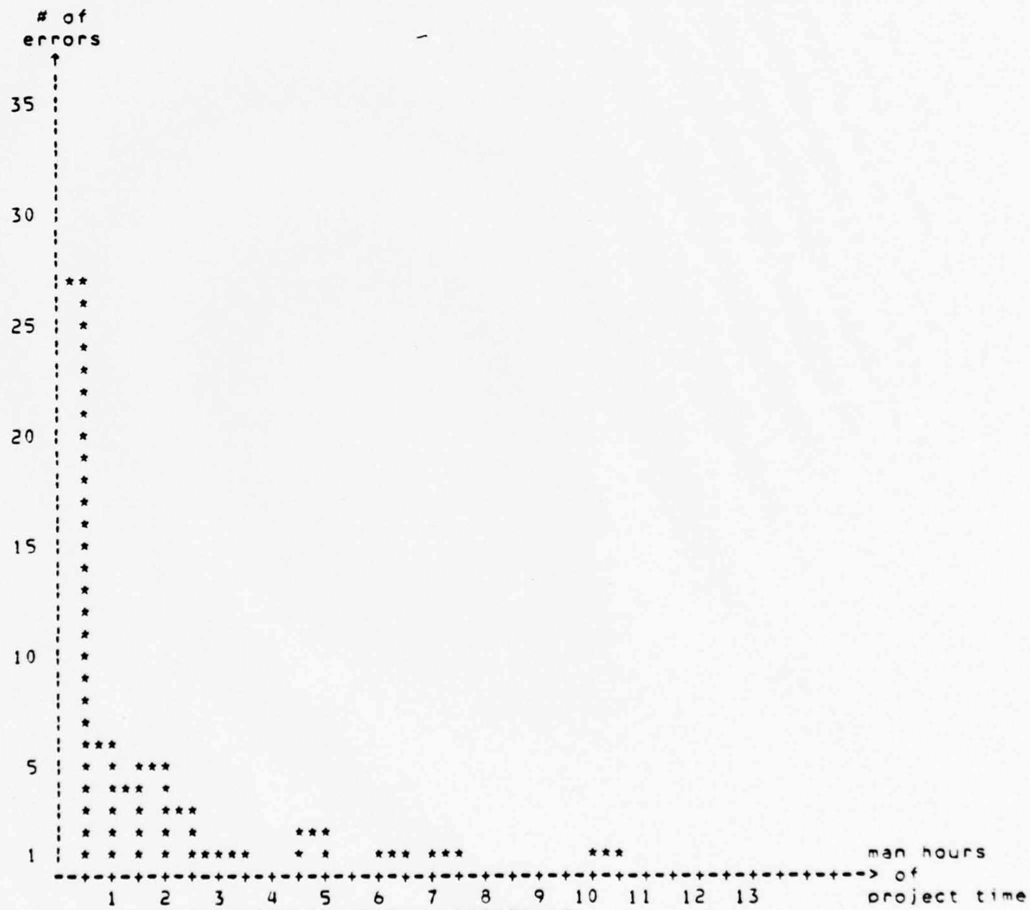
8. # of test and debug steps: 43

9. # of errors found: 50

10. Total man hours used to correct errors: 11.25

11. Error Detection:

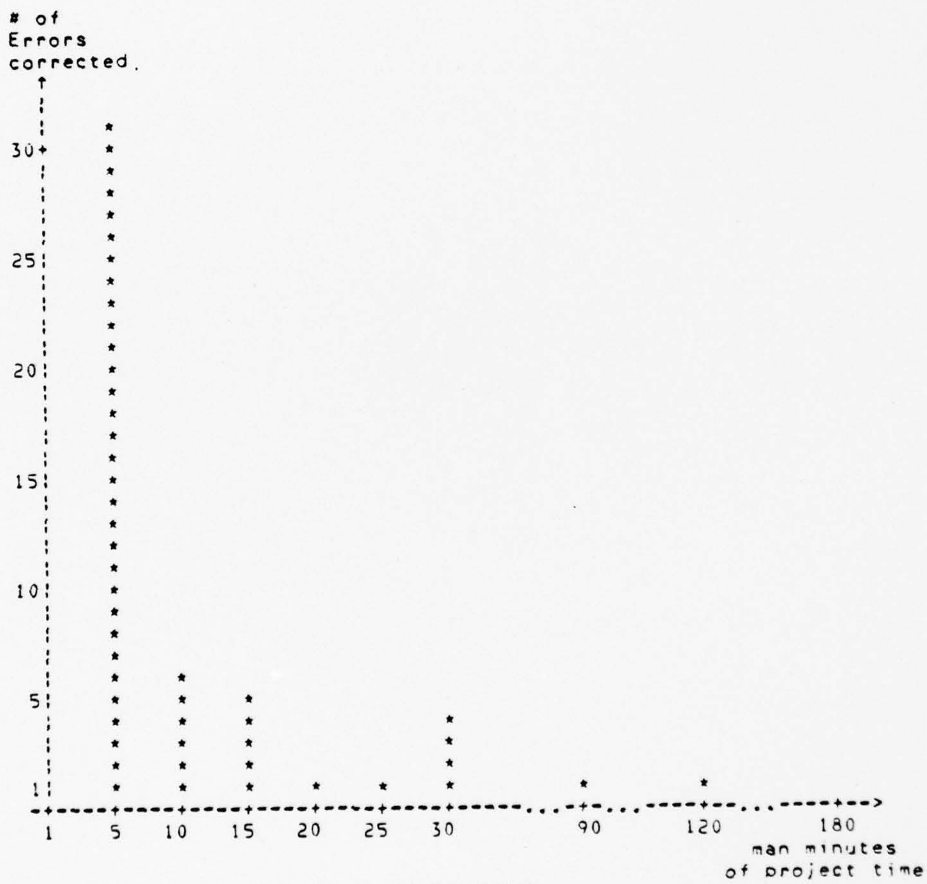
Mean time between error detections: 87.7 man min.



TIME BETWEEN ERROR DETECTIONS
(measured from point in time of detection of previous error)

12. Error Correction:

Mean time to correct an error: 13.3 man min.



TIME TO CORRECT ERRORS
(measured from point in time of detection)

13. When errors were found:

a) # of errors found during design phase:	0 =	0.0 %
b) # of errors found during design review:	0 =	0.0 %
c) # of errors found during coding:	4 =	8.0 %
d) # of errors found during debugging:	45 =	90.0 %
e) # of errors found during writing of test procedures:	0 =	0.0 %

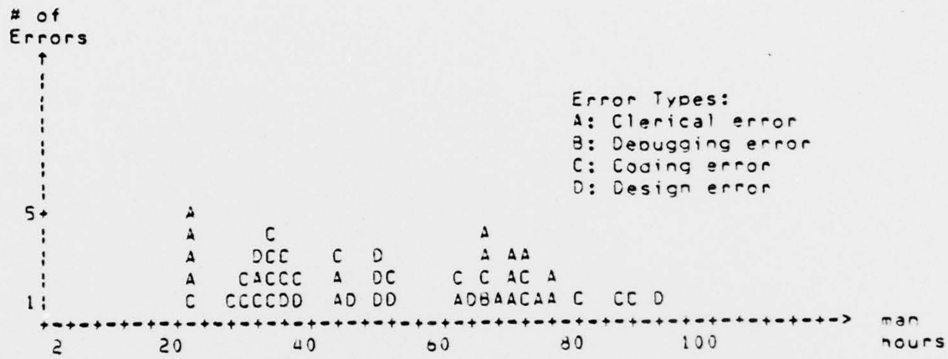
	50	
f) # of errors found during testing:	1 =	2.0 %

14. When errors were made:

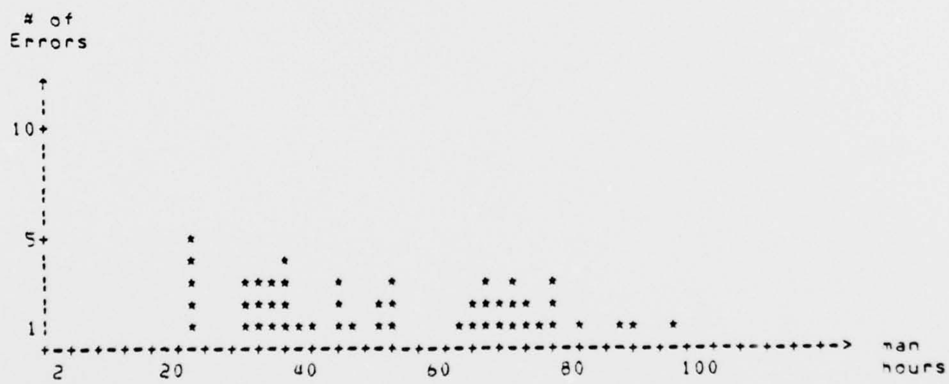
a) # of errors made during design phase:	10 =	20.0 %
b) # of errors made during design review:	0 =	0.0 %
c) # of errors made during coding:	37 =	74.0 %
d) # of errors made during debugging:	3 =	6.0 %
e) # of errors made during writing of test procedures:	0 =	0.0 %
f) # of errors made during testing:	0 =	0.0 %

	50	

15. TIME HISTORY GRAPHS :



NUMBER OF ERRORS FOUND VS PROJECT TIME



NUMBER OF ERRORS CORRECTED VS PROJECT TIME

1. Design Errors

The follwing types of errors apply to both categories "System Design Errors" and "Program Design Errors":

D1 : Communication Error
D2 : Design Negligence
D3 : Forgotten Cases or Steps
D4 : Timing Problems
D5 : Errors in I/O Concepts
D6 : Data Design Error
D7 : Initialization Error
D8 : Inadequate Checking
D9 : Extreme Conditions Neglected
D10: Sequencing Error
D11: Indexing Error
D12: Loop Control Errors
D13: Misuse of Boolean Expression
D14: Mathematical Error
D15: Representation Error
D16: Misunderstanding of Problem Specifications
D17: Other Design Errors

2. Coding Errors

C1 : Misunderstanding of Design
C2 : Negligence
C3 : I/O Format Error
C4 : Misplaced Data Declaration
C5 : Multiple Data Declarations
C6 : Missing Data Declaration
C7 : Inadequate Data
C8 : Initialization Error
C9 : Error in Parameter Passing
C10: Inadequate or Forgotten Checking
C11: Level Problems
C12: Missing Declarations of Block Limits
C13: Case selection error
C14: GO TO Problems
C15: Comment Error
C16: Forgotten Delimiter
C17: Inconsistency in Naming
C18: Wrong Use of Nested IF Statements
C19: Indexing Error
C20: Inconsistent Use of Variables or Data
C21: Sequencing Error
C22: Flag Usage Problems
C23: Syntax Error
C24: Loop Control Error
C25: Incorrect Exit from Subroutines
C26: Language Usage Problems

C27: Forgotten Statements
C28: Representation Error
C29: Control Sequence Error
C30: Incorrect Subroutine Usage
C31: Other Coding Errors

3. Clerical Errors

A1 : Manual Error
A2 : Mental Error
A3 : Procedural Errors
A4 : Other Clerical Errors

4. Debugging Errors

B1 : Inappropriate Use of Debugging Tools
B2 : Insufficient or Inappropriate Selection of Test Cases or Test Data
B3 : Misinterpretation of Debugging Results
B4 : Misinterpretation of Error Source
B5 : Negligence
B6 : Other Debugging Errors

5. Testing Errors

T1 : Inadequate Test Case(s) or Test Data
T2 : Misinterpretation of Test Results
T3 : Misinterpretation of Program Specification
T4 : Negligence
T5 : Other Testing Errors

DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : CLEARIN, CLEAROUT, CARDIN, PRINTOUT, TYPIN, TYPOUT, FILE I/O, TERMINAL I/O, GET OLD FILE INFORMATION, WRITE NEW FILE INFORMATION, CAR1, CAR2, CDR1, CDR2, SETCAR1, SETCAR2, SETCDR1, SETCDR2, FREE1, FREE2, FIND ID, GETQVALUE, MAIN

```
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 2  - 9
NUMBER OF NODES:     2
NUMBER OF ARCS :     1
NUMBER OF PATHS:     1
CYCLOMATIC NUMBER:  V(G)=     1

REACHABILITY OF NODES:
NODE     1  :     1
NODE     2  :     1
               --------
     SUM:      2.000000
REACHABILITY INDEX
OF DIRECTED GRAPH:        1.000000
```

NOTE: All 23 subroutines have the same structure.

DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

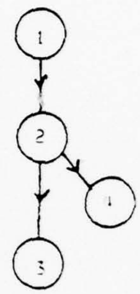
Program part : ALLOCATE1, ALLOCATE2, CONTROL

COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 8 - 9
NUMBER OF NODES: 4
NUMBER OF ARCS : 3
NUMBER OF PATHS: 2
CYCLOMATIC NUMBER: V(G)= 1

REACHABILITY OF NODES:
NODE 1 : 1
NODE 2 : 1
NODE 3 : 1
NODE 4 : 1

SUM: 4.000000
REACHABILITY INDEX
OF DIRECTED GRAPH: 1.000000



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : SPACE

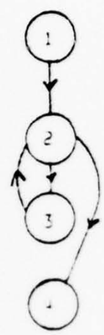
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 4

NUMBER OF NODES: 4
NUMBER OF ARCS : 4
NUMBER OF PATHS: 2
CYCLOMATIC NUMBER: V(G)= 2

REACHABILITY OF NODES:
NODE 1 : 1
NODE 2 : 2
NODE 3 : 1
NODE 4 : 2

SUM: 6.000000
REACHABILITY INDEX
OF DIRECTED GRAPH: 1.500000

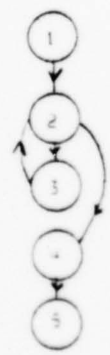


DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : INITIALIZE1, INITIALIZE2

```
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS:  7 - 9
NUMBER OF NODES:       5
NUMBER OF ARCS :       5
NUMBER OF PATHS:       2
CYCLOMATIC NUMBER:     V(G)=     2

REACHABILITY OF NODES:
NODE    1  :   1
NODE    2  :   2
NODE    3  :   1
NODE    4  :   2
NODE    5  :   2
              ---------
      SUM:     8.000000
REACHABILITY INDEX
OF DIRECTED GRAPH:         1.599999
```



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : INITIALIZE ALL

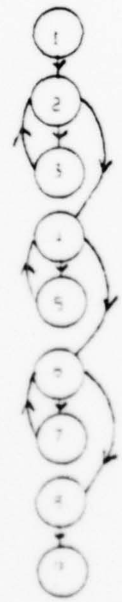
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 56
NUMBER OF NODES: 9
NUMBER OF ARCS : 11
NUMBER OF PATHS: 8
CYCLOMATIC NUMBER: V(G)= 4

REACHABILITY OF NCDES:

NCDE	1	:	1
NCDE	2	:	[illegible]
NCDE	3	:	[illegible]
NCDE	4	:	[illegible]
NCDE	5	:	[illegible]
NCDE	6	:	[illegible]
NCDE	7	:	[illegible]
NCDE	8	:	[illegible]
NCDE	9	:	[illegible]
SUM:			38.00000

REACHABILITY INDEX
OF DIRECTED GRAPH: 4.222221

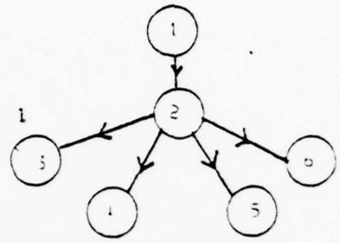


DIRECTED GRAPH REPRESENTATION

PROJECT = : 4

Program part : ERROR

```
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 24
NUMBER OF NODES:    6
NUMBER OF ARCS :    5
NUMBER OF PATHS:    4
CYCLOMATIC NUMBER:  V(G)=    1

REACHABILITY OF NODES:
NODE    1  :   1
NODE    2  :   1
NODE    3  :   1
NODE    4  :   1
NODE    5  :   1
NODE    6  :   1
              ---------
     SUM:      6.000000
REACHABILITY INDEX
OF DIRECTED GRAPH:       1.000000
```



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : DIAGNOSE

COMPLEXITY MEASURES:

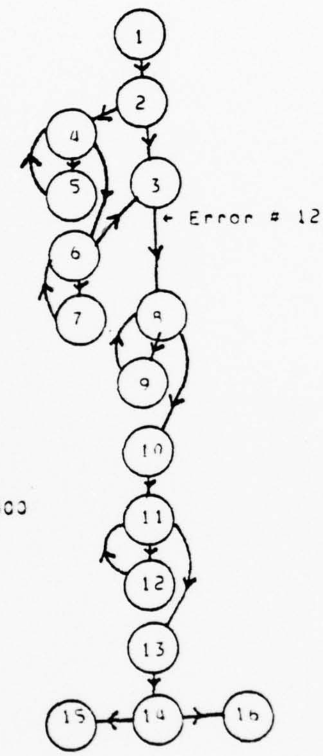
NUMBER OF STATEMENTS: 83

NUMBER OF NODES: 16
NUMBER OF ARCS : 20
NUMBER OF PATHS: 40
CYCLOMATIC NUMBER: V(G)= 6

REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	1
NODE	3 :	5
NODE	4 :	2
NODE	5 :	1
NODE	6 :	4
NODE	7 :	2
NODE	8 :	10
NODE	9 :	5
NODE	10 :	10
NODE	11 :	20
NODE	12 :	10
NODE	13 :	20
NODE	14 :	20
NODE	15 :	20
NODE	16 :	20
SUM:		151.0000

REACHABILITY INDEX
OF DIRECTED GRAPH: 9.437500



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : EXTRACT, FILL, LISTA, LISTC, LISTALL,
FIND VALUE LENGTH

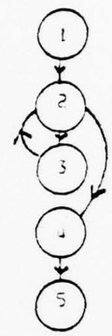
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 8 - 13

NUMBER OF NODES: 5
NUMBER OF ARCS : 5
NUMBER OF PATHS: 2
CYCLOMATIC NUMBER: V(G)= 2

REACHABILITY OF NODES:
NODE 1 : 1
NODE 2 : 2
NODE 3 : 1
NODE 4 : 2
NODE 5 : 2

SUM: 8.000000
REACHABILITY INDEX
OF DIRECTED GRAPH: 1.599999



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : GETCHAR

COMPLEXITY MEASURES:

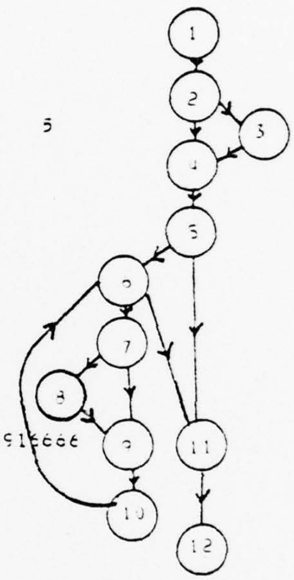
NUMBER OF STATEMENTS: 21

NUMBER OF NODES: 12
NUMBER OF ARCS : 15
NUMBER OF PATHS: 16
CYCLOMATIC NUMBER: V(G)= 5

REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	1
NODE	3 :	1
NODE	4 :	2
NODE	5 :	2
NODE	6 :	14
NODE	7 :	12
NODE	8 :	6
NODE	9 :	12
NODE	10 :	12
NODE	11 :	16
NODE	12 :	16
SUM:		95.00000

REACHABILITY INDEX
OF DIRECTED GRAPH: 7.916666



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : SUPER USER

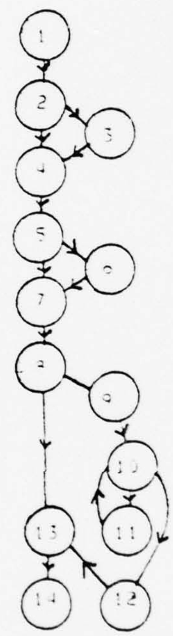
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 49
NUMBER OF NODES: 14
NUMBER OF ARCS : 17
NUMBER OF PATHS: 12
CYCLOMATIC NUMBER: V(G)= 5

REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	1
NODE	3 :	1
NODE	4 :	2
NODE	5 :	2
NODE	6 :	2
NODE	7 :	4
NODE	8 :	4
NODE	9 :	4
NODE	10 :	8
NODE	11 :	4
NODE	12 :	8
NODE	13 :	12
NODE	14 :	12

SUM: 65.00000
REACHABILITY INDEX
OF DIRECTED GRAPH: 4.642857

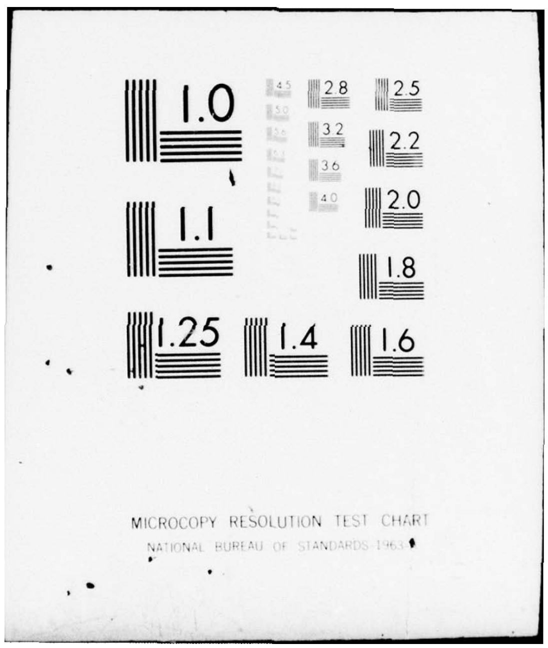


AD-A045 031 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
AN EXPERIMENT IN SOFTWARE ERROR OCCURRENCE AND DETECTION.(U)
JUN 77 H HOFFMANN

UNCLASSIFIED NL

4 OF 4
AD
A045031

END
DATE
FILMED
11 - 77
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963

DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : STORE

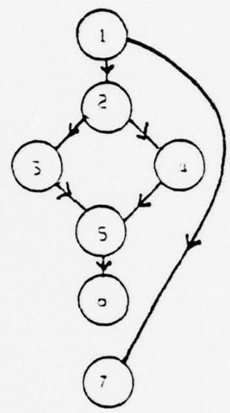
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 19
NUMBER OF NODES: 7
NUMBER OF ARCS: 8
NUMBER OF PATHS: 7
CYCLOMATIC NUMBER: V(G)= 3

REACHABILITY OF NODES:

NODE 1	:	7
NODE 2	:	6
NODE 3	:	3
NODE 4	:	3
NODE 5	:	6
NODE 6	:	6
NODE 7	:	7

SUM: 38.00000
REACHABILITY INDEX
OF DIRECTED GRAPH: 5.428571



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : GET ATTRIBUTES AND VALUES

a) # of nodes: 9

b) # of arcs : 11

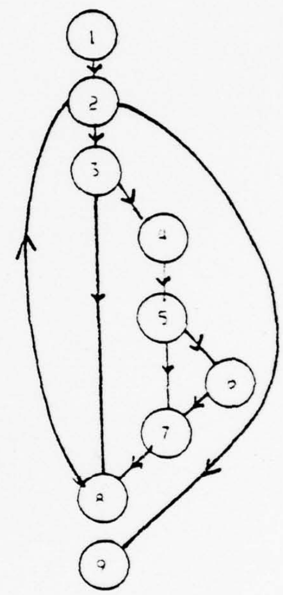
c) # of statements: 20

d) # of paths: *

e) Reachability: *

f) Cyclomatic number: 4

* Number of paths
and reachability
are very large.



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : GET ATTRIBUTE

COMPLEXITY MEASURES:

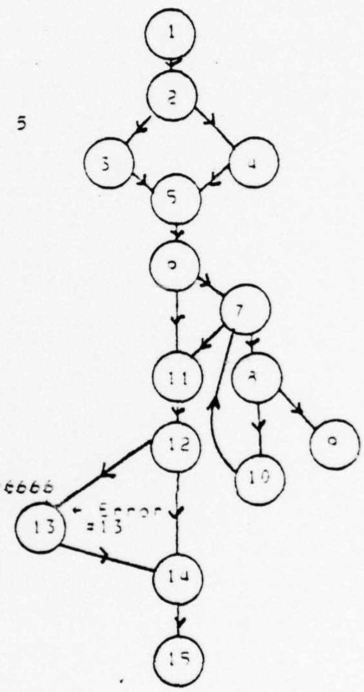
NUMBER OF STATEMENTS: 28

NUMBER OF NODES: 15
NUMBER OF ARCS : 18
NUMBER OF PATHS: 16
CYCLOMATIC NUMBER: V(G)= 5

REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	1
NODE	3 :	1
NODE	4 :	1
NODE	5 :	2
NODE	6 :	2
NODE	7 :	4
NODE	8 :	4
NODE	9 :	4
NODE	10 :	2
NODE	11 :	6
NODE	12 :	6
NODE	13 :	6
NODE	14 :	12
NODE	15 :	12
SUM:		64.00000

REACHABILITY INDEX
OF DIRECTED GRAPH: 4.266666



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : GET VALUE

a) # of nodes: 17

b) # of arcs : 23

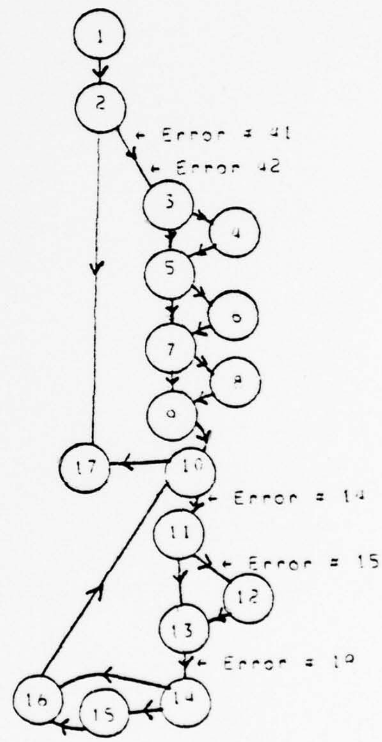
c) # of statements: 37

d) # of paths: *

e) Reachability: *

f) Cyclomatic number: 8

* Number of paths and reachability are very large.



DIRECTED GRAPH REPRESENTATION

PROJECT = : 4

Program part : FIND ATTRIBUTE

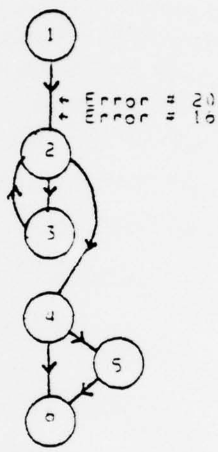
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 13

NUMBER OF NODES: 6
NUMBER OF ARCS : 7
NUMBER OF PATHS: 4
CYCLOMATIC NUMBER: V(G)= 3

REACHABILITY OF NODES:
NODE 1 : 1
NODE 2 : 2
NODE 3 : 1
NODE 4 : 2
NODE 5 : 2
NODE 6 : 4

SUM: 12.00000
REACHABILITY INDEX
OF DIRECTED GRAPH: 2.000000



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : SKIP, FIND COMMAND, GET CONDITIONS,
TEST ALL

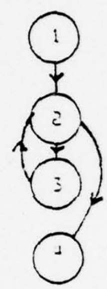
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 5 - 12
NUMBER OF NODES: 4
NUMBER OF ARCS : 4
NUMBER OF PATHS: 2
CYCLOMATIC NUMBER: V(G)= 2

REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	2
NODE	3 :	1
NODE	4 :	2
SUM:		6.000000

REACHABILITY INDEX
OF DIRECTED GRAPH: 1.500000



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : INTERPRET

COMPLEXITY MEASURES:

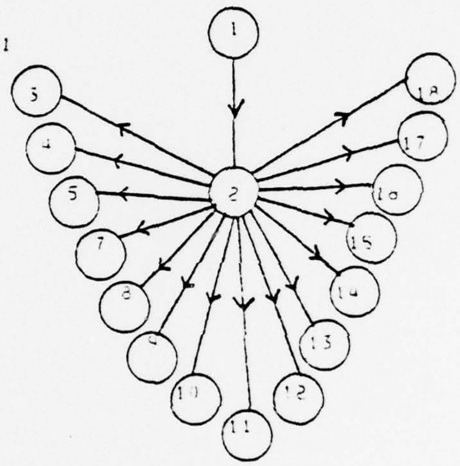
NUMBER OF STATEMENTS: 21

NUMBER OF NODES: 18
NUMBER OF ARCS : 17
NUMBER OF PATHS: 16
CYCLOMATIC NUMBER:
V(G) = 1

REACHABILITY OF NODES:

NODE	1	:	1
NODE	2	:	1
NODE	3	:	1
NODE	4	:	1
NODE	5	:	1
NODE	6	:	1
NODE	7	:	1
NODE	8	:	1
NODE	9	:	1
NODE	10	:	1
NODE	11	:	1
NODE	12	:	1
NODE	13	:	1
NODE	14	:	1
NODE	15	:	1
NODE	16	:	1
NODE	17	:	1
NODE	18	:	1

SUM: 18.00000
REACHABILITY INDEX
OF DIRECTED GRAPH: 1.000000



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : SHIFT

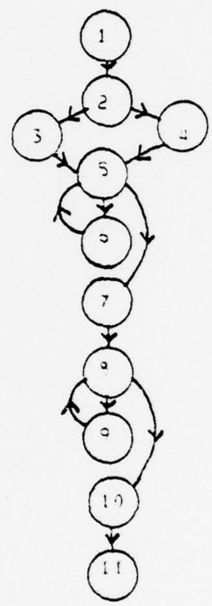
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 21
NUMBER OF NODES: 11
NUMBER OF ARCS : 13
NUMBER OF PATHS: 8
CYCLOMATIC NUMBER: V(G)= 4

REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	1
NODE	3 :	1
NODE	4 :	1
NODE	5 :	4
NODE	6 :	2
NODE	7 :	4
NODE	8 :	8
NODE	9 :	4
NODE	10 :	8
NODE	11 :	8
SUM:		42.00000

REACHABILITY INDEX
OF DIRECTED GRAPH: 3.818181



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : LISTPHASE

COMPLEXITY MEASURES:

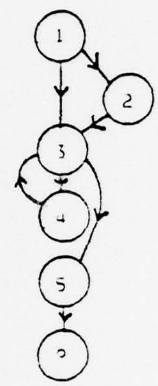
NUMBER OF STATEMENTS: 11

NUMBER OF NODES: 6
NUMBER OF ARCS : 7
NUMBER OF PATHS: 4
CYCLOMATIC NUMBER: V(G)= 3

REACHABILITY OF NODES:

NODE 1	:	1
NODE 2	:	1
NODE 3	:	4
NODE 4	:	2
NODE 5	:	4
NODE 6	:	4
SUM:		16.00000

REACHABILITY INDEX
OF DIRECTED GRAPH: 2.666666

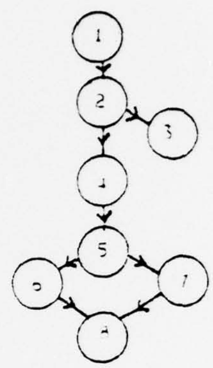


DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : LISTSHORT

```
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 13
NUMBER OF NODES:      8
NUMBER OF ARCS :      8
NUMBER OF PATHS:      3
CYCLOMATIC NUMBER:  V(G)=    2

REACHABILITY OF NODES:
NODE     1  :    1
NODE     2  :    1
NODE     3  :    1
NODE     4  :    1
NODE     5  :    1
NODE     6  :    1
NODE     7  :    1
NODE     8  :    2
               ---------
     SUM:       9.000000
REACHABILITY INDEX
OF DIRECTED GRAPH:          1.125000
```



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : FIND MEMBER, FINDATTRIBUTE, WRITE RESULTS

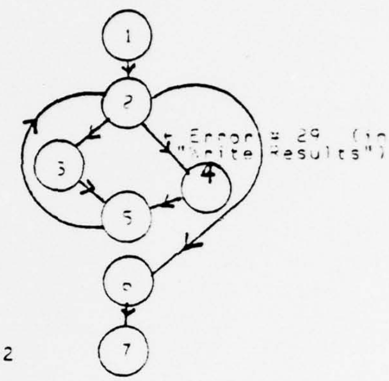
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 13 - 16
NUMBER OF NODES: 7
NUMBER OF ARCS : 8
NUMBER OF PATHS: 7
CYCLOMATIC NUMBER: V(G)= 3

REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	7
NODE	3 :	3
NODE	4 :	3
NODE	5 :	6
NODE	6 :	7
NODE	7 :	7
SUM:		34.00000

REACHABILITY INDEX
OF DIRECTED GRAPH: 4.857142



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : LISTATTRIBUTE

a) # of nodes: 16

b) # of arcs : 21

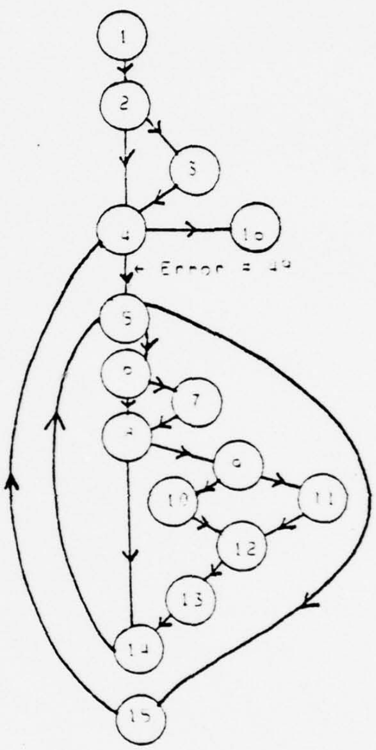
c) # of statements: 34

d) # of paths: *

e) Reachability: *

f) Cyclomatic number: 7

* Number of paths
and reachability
are very large.

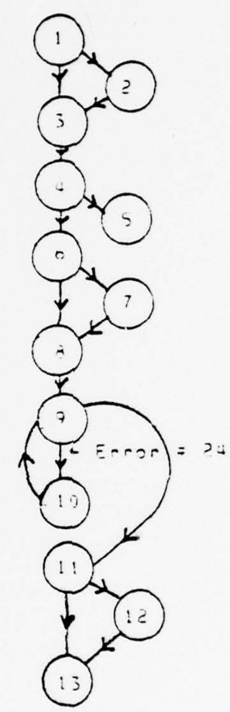


. DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : LISTM

```
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 24
NUMBER OF NODES:    13
NUMBER OF ARCS :    16
NUMBER OF PATHS:    13
CYCLOMATIC NUMBER:  V(G)=    5

REACHABILITY OF NODES:
NODE     1  :    1
NODE     2  :    1
NODE     3  :    2
NODE     4  :    2
NODE     5  :    2
NODE     6  :    2
NODE     7  :    2
NODE     8  :    4
NODE     9  :    8
NODE    10  :    4
NODE    11  :    8
NODE    12  :    8
NODE    13  :   16
                ---------
     SUM:       60.00000
REACHABILITY INDEX
OF DIRECTED GRAPH:          4.615384
```



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : COMPARE RANGE

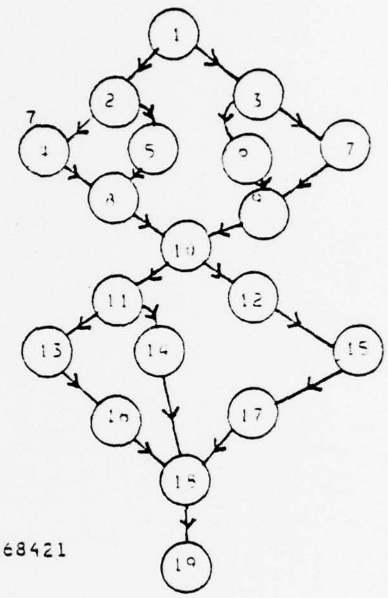
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 19
NUMBER OF NODES: 19
NUMBER OF ARCS: 24
NUMBER OF PATHS: 16
CYCLOMATIC NUMBER: V(G)= 7

REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	1
NODE	3 :	1
NODE	4 :	1
NODE	5 :	1
NODE	6 :	1
NODE	7 :	1
NODE	8 :	2
NODE	9 :	2
NODE	10 :	4
NODE	11 :	4
NODE	12 :	4
NODE	13 :	4
NODE	14 :	4
NODE	15 :	4
NODE	16 :	8
NODE	17 :	8
NODE	18 :	16
NODE	19 :	16

SUM: 83.00000
REACHABILITY INDEX
OF DIRECTED GRAPH: 4.368421



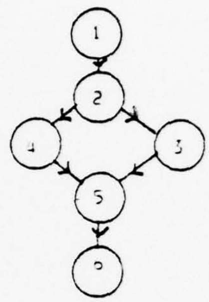
DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : ATTR, MATCHING, CHANGE KEY

COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 10 - 17
NUMBER OF NODES: 6
NUMBER OF ARCS : 6
NUMBER OF PATHS: 2
CYCLOMATIC NUMBER: V(G)= 2



REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	1
NODE	3 :	1
NODE	4 :	1
NODE	5 :	2
NODE	6 :	2

SUM: 8.000000
REACHABILITY INDEX
OF DIRECTED GRAPH: 1.333333

DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : FA

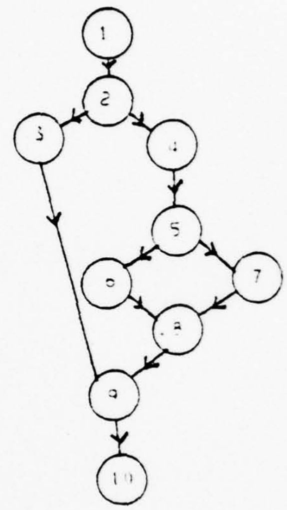
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 19
NUMBER OF NODES: 10
NUMBER OF ARCS : 11
NUMBER OF PATHS: 3
CYCLOMATIC NUMBER: V(G) = 3

REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	1
NODE	3 :	1
NODE	4 :	1
NODE	5 :	1
NODE	6 :	1
NODE	7 :	1
NODE	8 :	2
NODE	9 :	3
NODE	10 :	3
SUM:		15.00000

REACHABILITY INDEX
OF DIRECTED GRAPH: 1.500000



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : FIND

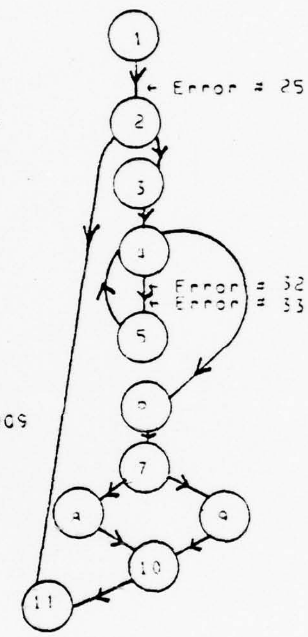
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 18
NUMBER OF NODES: 11
NUMBER OF ARCS : 13
NUMBER OF PATHS: 5
CYCLOMATIC NUMBER: V(G)= 4

REACHABILITY OF NODES:

NODE 1	:	1
NODE 2	:	1
NODE 3	:	1
NODE 4	:	2
NODE 5	:	1
NODE 6	:	2
NODE 7	:	2
NODE 8	:	2
NODE 9	:	2
NODE 10	:	4
NODE 11	:	5
SUM:		23.00000

REACHABILITY INDEX
OF DIRECTED GRAPH: 2.090909



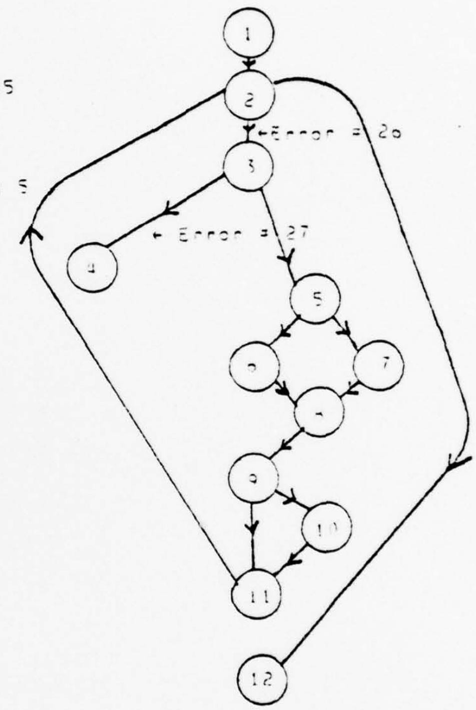
DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : GET REQUEST

a) # of nodes: 12
b) # of arcs : 15
c) # of statements: 35
d) # of paths: *
e) Reachability: *
f) Cyclomatic number: 5

* Number of paths and reachability are very large.



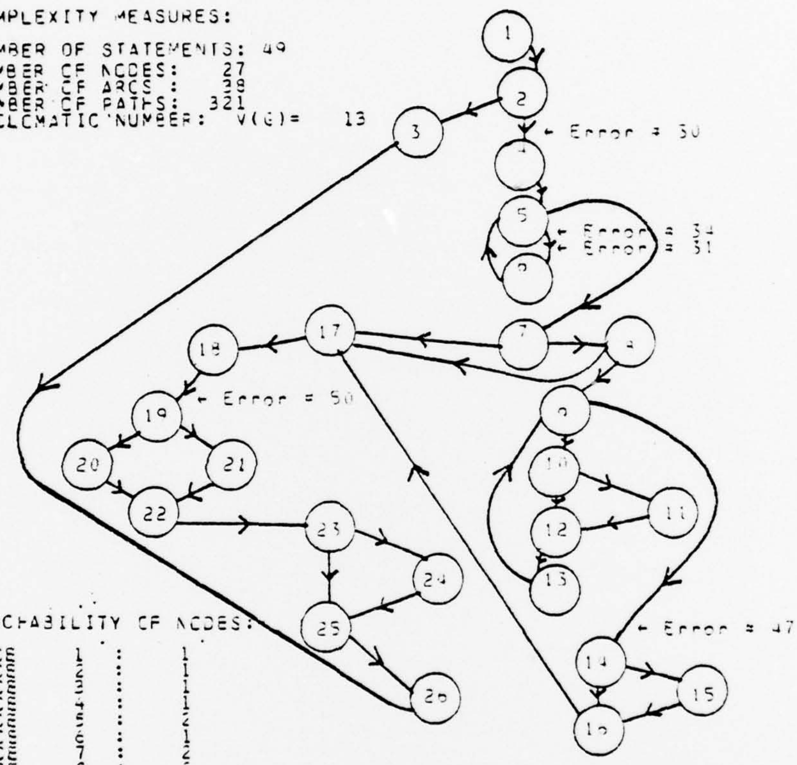
DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : EQUAL

COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 49
NUMBER OF NODES: 27
NUMBER OF ARCS : 38
NUMBER OF PATHS: 321
CYCLOMATIC NUMBER: V(G)= 13



REACHABILITY OF NODES:

NODE	1	:	1
NODE	2	:	1
NODE	3	:	1
NODE	4	:	1
NODE	5	:	2
NODE	6	:	1
NODE	7	:	2
NODE	8	:	2
NODE	9	:	14
NODE	10	:	12
NODE	11	:	6
NODE	12	:	12
NODE	13	:	12
NODE	14	:	14
NODE	15	:	14
NODE	16	:	28
NODE	17	:	32
NODE	17	:	0
NODE	18	:	32
NODE	19	:	64
NODE	20	:	64
NODE	21	:	64
NODE	22	:	128
NODE	23	:	160
NODE	24	:	160
NODE	25	:	320
NODE	26	:	321

SUM: 1468.000
REACHABILITY INDEX
OF DIRECTED GRAPH: 54.37036

DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : TEST CONDITIONS

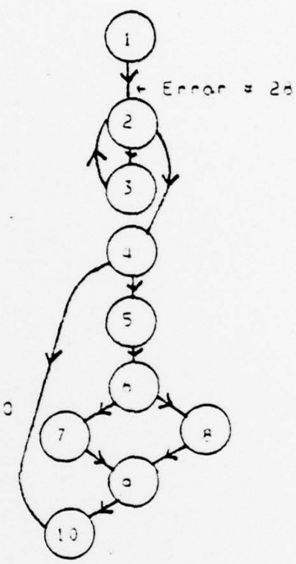
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 19
NUMBER OF NODES: 10
NUMBER OF ARCS : 12
NUMBER OF PATHS: 6
CYCLOMATIC NUMBER: V(G)= 4

REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	2
NODE	3 :	1
NODE	4 :	2
NODE	5 :	2
NODE	6 :	2
NODE	7 :	2
NODE	8 :	2
NODE	9 :	4
NODE	10 :	6
SUM:		24.00000

REACHABILITY INDEX
OF DIRECTED GRAPH: 2.400000



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : COMPARE MANY

a) # of nodes: 12

b) # of arcs : 14

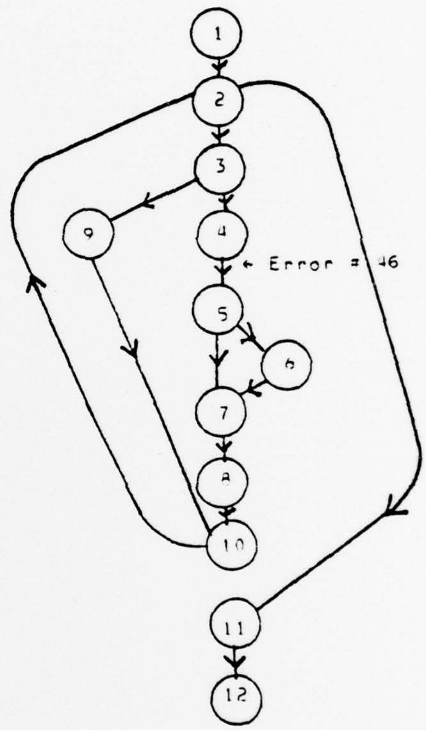
c) # of statements: 27

d) # of paths: *

e) Reachability: *

f) Cyclomatic number: 4

* Number of paths and reachability are very large.



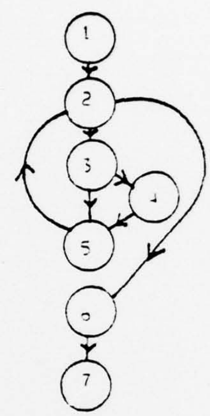
DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : LLIMIT

COMPLEXITY MEASURES:

```
NUMBER OF STATEMENTS: 15
NUMBER OF NODES:      7
NUMBER OF ARCS :      8
NUMBER OF PATHS:      7
CYCLOMATIC NUMBER:    V(G)=     3

REACHABILITY OF NODES:
NODE     1   :    1
NODE     2   :    7
NODE     3   :    5
NODE     4   :    3
NODE     5   :    6
NODE     6   :    7
NODE     7   :    7
                --------
    SUM:         36.00000
REACHABILITY INDEX
OF DIRECTED GRAPH:        5.142857
```



DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : ULIMIT

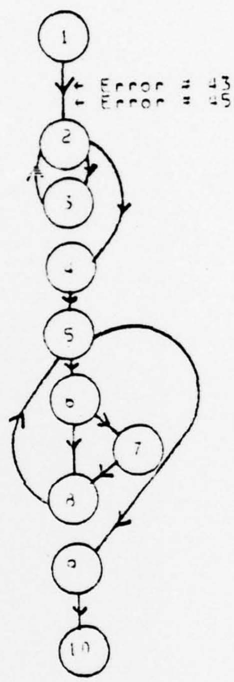
COMPLEXITY MEASURES:

NUMBER OF STATEMENTS: 17
NUMBER OF NODES: 10
NUMBER OF ARCS : 12
NUMBER OF PATHS: 14
CYCLOMATIC NUMBER: V(G)= 4

REACHABILITY OF NODES:

NODE	1 :	1
NODE	2 :	2
NODE	3 :	1
NODE	4 :	2
NODE	5 :	14
NODE	6 :	10
NODE	7 :	6
NODE	8 :	12
NODE	9 :	14
NODE	10 :	14
SUM:		76.00000

REACHABILITY INDEX
OF DIRECTED GRAPH: 7.599999

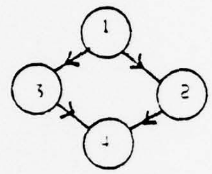


DIRECTED GRAPH REPRESENTATION

PROJECT # : 4

Program part : SWITCH

COMPLEXITY MEASURES:




```
NUMBER OF STATEMENTS: 15
NUMBER OF NODES:      4
NUMBER OF ARCS :      4
NUMBER OF PATHS:      2
CYCLOMATIC NUMBER:    V(G)=     2

REACHABILITY OF NODES:
NODE    1  :   1
NODE    2  :   1
NODE    3  :   1
NODE    4  :   2
              ---------
   SUM:        5.000000
REACHABILITY INDEX
OF DIRECTED GRAPH:        1.250000
```

TEST PHASE DESCRIPTION

Project # : 4

Test run # : 1 Including 5 Test Steps

Begin of Test (day/time) : 04/29/1330 End of Test (day/time) : 04/29/1530

CPU time for necessary compiles (in sec.): 9.09
a) 9.09 b) c) d) e) f) g)

CPU time for TEST run (sec) : 12.8

Man Hours for this Test run : 2.0 (including preparation of tests)

TEST STEP	OBJECTVE	EXPECTED RESULT (TOLERANCE)	ACTUAL RESULT	ERROR # [1)]	DAY TIME	COMMENTS AND CODED ERROR TYPES
1	Test of I/O functions under CP/CMS	I/O can be directed as designed: -Data base information is read from file labeled "DBASE INPUT" -Commands are input via terminal only -Output can be directed by user to terminal or file	O.K.		4/29 1330	1) Record when error occurs
2	Check boundary conditions under CP/CMS	Program neglects input or provides appropriate error messages	O.K.			
3	Test SU functions under CP/CMS	a)Members can be added to the data base b)Data base information can be can be written on output file such that this file could be used as input file for subsequent runs c)Items of M2 can be examined	O.K.			
4	Check Diagnostics		O.K.			
5	Test Termination of program		O.K.		4/29 1550	

TEST PHASE DESCRIPTION

Project # : 4

Test run # : 2 Including 2 Test Steps

Begin of Test (day/time) : 05/01/1200 End of Test (day/time) : 05/02/2200

CPU time for necessary compiles (in sec.): 8.75
a) 8.75 b) c) d) e) f) g)

CPU time for TEST run (sec) : 12.40

Man Hours for this Test run : 4.0 (including preparation of tests)

TEST STEP	OBJECTVE	EXPECTED RESULT (TOLERANCE)	ACTUAL RESULT	ERROR # [1]	DAY TIME	COMMENTS AND CODED ERROR TYPES
1	Test comparison of range values	Comparison must always be correct	Does not work for nega-tive num-bers		5/01 1200 1700 5/01 1730	1) Record when error occurs D3 Implementation of new code necessary to handle all cases
2	Repeat step 1 using corrected version of program		O.K.		5/02 1600 5/02 2200	

TEST PHASE DESCRIPTION

Project # : 4

Test run # : 3   Including 1 Test Steps

Begin of Test (day/time) : 05/03/0900   End of Test (day/time) : 05/03/1800

CPU time for necessary compiles (in sec.): 9.68
a) 9.68 b) c) d) e) f) g)

CPU time for TEST run (sec) : 12.84

Man Hours for this Test run : 5.0 (including preparation of tests)

TEST STEP	OBJECTVE	EXPECTED PESULT (TOLERANCE)	ACTUAL RESULT	1) ERROR #	DAY TIME	COMMENTS AND CODED ERROR TYPES
1	Test program under CP/CMS using radars as data members	All functions should work the same way as for members used in previous test and debug runs (provided that all attribute value pairs are within designed limits)	O.K.		5/03 0900 5/03 1800	1) Record when error occurs

BIBLIOGRAPHY

1. Yourdon, Edward, "Measuring the Goodness of Computer Programs", ACPA Thruput, April 1972, pp 2-7.

2. AUERBACH "Computer Programming Management Methodology", 14-01-01, 1975 p 2.

3. McDaniel, Herman, US Civil Service Commission, "An Introduction to Decision Logic Tables", John Wiley & Sons, Inc. 1968, p 52.

4. Packer, David, W., "Effective Program Design", Computers and Automation for July 1970, p 38.

5. AUERBACH "Computer Programming Management Methodology", 14-02-05, 1975 p 3.

6. Freeman, Peter, "Improving the Review of Software Designs", Volume 44, AFIPS Press, 1975.

7. Boehm, Barry W. , "Software Engineering", IEEE Transactions on Computers, Vol. C-25, No. 12, Dec. 1976.

8. Boehm, C. & Jacopini, G., "Flow-Diagrams, Turing Machines and Languages With Only Two Formation Rules", Comm. ACM, May, 1966 pp 366-371.

9. AUERBACH "Computer Programming Management Methodology", 14-04-01, 1974 p 3.

10. Endres, Albert, "An Analysis of Errors and their Causes in System Programs", IEEE Transactions on Software Engineering, Vol. SE-1, No. 2, June 1975.

11. AUERBACH "Computer Programming Management Methodology", 14-02-06, 1976 pp 5-11.

12. Yourdon, Edward, "Techniques of Program Structure and Design", Prentice-Hall, Inc., p 248.

13. Fisher, Sir Ronald A., "The Design of Experiments", Hafner Publishing Company, New York 1971.

14. Baker, F. T., "Chief Programmer Team Management of Production Programming", IBM System Journal, Vol. 11, No. 1, 1972.

15. Dolotta T. A. et al., "Data Processing in 1980-85", New York, Wiley Interscience, 1976.

16. Bradley G. H., Howard G. T., Schneidewind N. F., Montgomery G. W., and Green T. F., "System Test Methodology", Naval Postgraduate School, Vol. I and Vol. II, July 1975.

17. Schneidewind N. F., "An Approach To Software Reliability Prediction and Quality Control", AFIPS Conference Proceedings, v. 41, Part II, Fall, Joint Computer Conference, pp 837-838, 1972.

18. Boehm, B. W., "Software and Its Impact: A Quantitative Assesment", Datamation, pp 48-59, May 1973.

19. Boehm B. W., McLean R. K., and Urfig D. B., "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software", Proceeding International Conference on Reliable Software, pp 105-113, 1975.

20. Trivedi A. K., and Shooman M. L., "A Many-State Markov Model for the Estimation and Prediction of Computer Software Performance Parameters", Proceeding International Conference on Reliable Software, pp 208-215, 1975.

21. Rubey R. J., "Quantitative Aspects of Software Validation", Proceeding International Conference on Reliable Software, p 246-251, 1975.

22. Schneidewind N. F., "Analysis of Error Processes in Computer Software", Proceeding International Conference on Reliable Software, pp 337-346, 1975.

23. Shooman M. L., and Bolsky M. I., "Types and Distribution and Test and Correction Times for Programming Errors", Proceeding International Conference on Reliable Software, pp 347-357, 1975.

24. Goodenough J. B., and Gerhart S. L., "Towards A Theory of Test Data Selection", Proceeding International Conference on Reliable Software, pp 493-510, 1975.

25. Weinberg G. M., "Pschychology of Computer Programming", Van Nostrand Reinhold Company, 1971.

26. Hetzel W. C., "Program Test Methods", Prentice-Hall, Inc., 1973.

27. AUERBACH, "Computer Programmming Management Methodology", 14-02-01, 1974 p 6.

28. Kirchgaessner M., "Analysis of Program Structure and Error Characteristics As Applied to NTDS Programs", Master of Science Thesis, Naval Postgraduate School, Monterey CA., 1976.

29. Randell B. and Buxton J. N., "Software Engineering Techniques", NATO Scientific Affairs, Brussels 39, Beligium 1970, p 21.

30. Dijkstra E. W., "Programming Considered as a Human Activity", Proceedings of IFIP Congress 65, Spartan Books, Washington, D.C., 1965.

31. Dijkstra E. W., "GO-TO Statement Considered Harmful", Letter to the Editor, Communications of ACM, March 1968.

32. McCabe T. J., "A Complexity Measure", IEEE Transactions on Software Engineering, Dec. 1976, pp 308-320.

33. Schneidewind N. F., Howard G. T., Kirchgaessner M., "Software Error Detection Models, Validation Tests and Program Complexity Measures", Naval Postgraduate School, Monterey CA., November 1976.

34. Schneidewind N. F., Hoffmann H. M., "Software Structure and Error Properties: Models vs. Real Programs", Naval Postgraduate School, Monterey CA., April 1977.

35. Abramson Norman, "Information Theory and Coding", McGraw Hill, 1963, pp 163-165.

36. UNIVAC, "User's Reference Manual for Compiler Monitor System", Vol. II, Nov. 1971, p II-5-44.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
4.	Professor Norman F. Schneidewind, Code 52SS Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
5.	LT G.M. Raetz, Code 52RR Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
6.	Marineamt -A 1- 2940 Wilhelmshaven Federal Republic of Germany	1
7.	Dokumentationszentrale der Bundeswehr (See) Friedrich-Ebert-Allee 34 5300 Bonn Federal Republic of Germany	1
8.	Kommando Marinefuehrungssysteme 4. Einfahrt 2940 Wilhelmshaven Federal Republic of Germany	1
9.	MFmStab 70 2390 Flensburg-Muerwik Federal Republic of Germany	1

10. Mr. Richard Pariseau 1
Naval Air Development Center
Warminster, Pennsylvania 18174

11. LCDR Heinz-Michael Hoffmann 1
MFmStab 70
2390 Flensburg-Muerwik
Federal Republic of Germany